This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2833478, IEEE Transactions on Knowledge and Data Engineering



# Efficient Vertical Mining of High Average-Utility Itemsets based on Novel Upper-Bounds

Tin Truong, Hai Duong, Bac Le, Philippe Fournier-Viger

**Abstract**—Mining High Average-Utility Itemsets (HAUIs) in a quantitative database is an extension of the traditional problem of frequent itemset mining, having several practical applications. Discovering HAUIs is more challenging than mining frequent itemsets using the traditional support model since the average-utilities of itemsets do not satisfy the downward-closure property. To design algorithms for mining HAUIs that reduce the search space of itemsets, prior studies have proposed various upper-bounds on the average-utilities of itemsets. However, these algorithms can generate a huge amount of unpromising HAUI candidates, which result in high memory consumption and long runtimes. To address this problem, this paper proposes four tight average-utility upper-bounds, based on a vertical database representation, and three efficient pruning strategies. Furthermore, a novel generic framework for comparing average-utility upper-bounds is presented. Based on these theoretical results, an efficient algorithm named dHAUIM is introduced for mining the complete set of HAUIs. dHAUIM represents the search space and quickly compute upper-bounds using a novel IDUL structure. Extensive experiments show that dHAUIM outperforms three state-of-the-art algorithms for mining HAUIs in terms of runtime on both real-life and synthetic databases. Moreover, results show that the proposed pruning strategies dramatically reduce the number of candidate HAUIs.

**Index Terms**—Pattern mining, utility mining, high average-utility itemset, upper-bound, diffset

## 1 Introduction

THE problem of mining frequent itemsets and association rules in transaction databases [1] is a fundamental research problem in the field of knowledge discovery in databases, which consists of discovering frequently purchased itemsets (sets of items) in customer transactions. To efficiently solve this problem for binary transaction databases, algorithms have been developed based on the downward closure property (DCP) of the support measure for itemsets. This property indicates that, if an itemset is infrequent, all its super itemsets are also infrequent, and can thus be pruned to reduce the search space. A popular extension of frequent itemset mining is weighted itemset mining, where weights are assigned to items to indicate their relative importance. The goal is to discover itemsets having weights that are no less than a user-defined threshold [2], [3], [4]. Although these prior studies have several applications, real-life transactional databases usually not only contain weights (e.g. unit profits of items) but also information about the quantities of items in transactions (e.g. purchase quantities). But items having high selling frequencies (support) are often not the most important to the user. For example, many items may have high selling frequencies but may yield a low profit, while less frequent items may yield a higher profit. Thus, only considering the frequencies or weights of itemsets is insufficient to identify itemsets of interest to users.

Facing these limitations of traditional and weighted frequent itemset mining, the problem of high utility itemset mining (HUIM) has emerged as an important research problem [5], [6], [7], [8], [9]. The goal is to extract itemsets that have a high utility (yield a high profit), where the utility of an itemset is defined as the sum of the utilities of its items in transactions where the itemset appears. An itemset is of high utility (HU) if its utility is larger than or equal to a minimum utility threshold *mu*, defined by the user. The key challenge for designing efficient HUIM algorithms is that the DCP does not hold for the utility measure. To address this issue, an upper bound (UB) on the utility of itemsets that respect the DCP was proposed [6], named transaction-weighted utilization (TWU). The DCP property of the TWU upper-bound states that supersets of an itemset having a TWU value that is less than the *mu* threshold also have low TWU values, and are low utility itemsets. The TWU upper-bound has been used in numerous HUIM algorithms to prune unpromising itemsets and thus reduce the search space [10], [11].

Although high utility itemset mining is useful, the standard utility measure does not take the length of itemsets into account. But in real-life, an itemset containing many items is often uninteresting as it is difficult to co-promote a large set of products. Besides, itemsets containing several items often have a higher utility than those containing fewer items, which may be seen as unfair. For instance, many high utility items such as laptop produce high utility itemsets when they are combined with items having a lower utility such as USB drives or mouse pads. This can result


- *T. Truong is with the Department of Mathematics and Computer Science, University of Dalat, 1 Phu Dong Thien Vuong Street, Dalat city, Vietnam. E-mail: tintc@dlu.edu.vn.*
- *H. Duong is with the Department of Mathematics and Computer Science, University of Dalat, 1 Phu Dong Thien Vuong Street, Dalat city, Vietnam. E-mail: haidv@dlu.edu.vn.*
- *B. Le is with the VNU – HCMC, University of Natural Sciences, 227 Nguyen Van Cu Street, District 5, Ho Chi Minh city, Vietnam. E-mail: lhbac@fit.hcmus.edu.vn.*
- *P. Fournier-Viger is with the School of Humanities and Social Sciences, Harbin Institute of Technology (Shenzhen), HIT Campus, Pingshan Rd, University Town, Xili, Shenzhen, China. E-mail: philfv8@yahoo.com.*




xxxx-xxxx/0x/$xx.00 © 200x IEEE Published by the IEEE Computer Society

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2833478, IEEE Transactions on Knowledge and Data Engineering



in a huge number of high utility patterns containing information that is not useful for decision-making. To more fairly measure the utility of itemsets, Hong et al. have proposed the average utility measure [12], [13], which adds a penalty that is proportional to the length of an itemset to the utility measure. The average-utility of an itemset is the sum of the utilities of its items in transactions where it appears, divided by its length. Although, high average-utility itemset mining (HAUIM) is a well-studied research problem, extracting HAUIs remains time-consuming. Thus, proposing techniques to improve the efficiency of HAUIM is important.

The key challenge for designing efficient HAUIM algorithms is that the DCP does not hold for the average-utility (AU) of itemsets (similarly to HUIM). To deal with this issue, researchers have proposed an average-utility upper-bound (UB) on the average-utilities of itemsets named *auub* [14], [15], which restores the DCP. Then, to further reduce the search space, an improved upper-bound has been designed and integrated in a projection based two-phase algorithm [16]. In the first phase, a set of candidate high utility itemsets is found using the improved upper-bound. In the second phase, the average utilities of candidates are calculated to filter low average-utility itemsets and obtain the set of HAUIs. However, repeatedly scanning projected databases and updating UBs are time-consuming. To more efficiently mine HAUIs, an AU-list structure and a corresponding pruning strategy were proposed in the HAUI-Miner algorithm [17]. Recently, Lin et al. [18], [19] and Yun et al. [20] proposed additional alternative UBs to enhance the performance of HAUIM. Despite these advances, UBs used by current state-of-the-art HAUIM algorithms remains loose. Thus, algorithms cannot reduce the search space effectively, consider a large amount of unpromising candidate itemsets, and as a result can have long runtimes. Moreover, although many UBs have been proposed, no generic theoretical model has yet been designed to compare their ability at pruning unpromising candidates. Based on these observations, three important research questions are raised:

($\boldsymbol{Q_1}$) Can we propose better UBs?

($\boldsymbol{Q_2}$) Can we design a generic framework to evaluate and compare the ability at reducing the search space of UBs? Can this framework help draw insights on how to design suitable pruning strategies for new UBs?

($\boldsymbol{Q_3}$) Can we design a suitable structure to maintain auxiliary utility information, a fast UB calculation method and an efficient algorithm to decrease the runtime of HAUIM?

This study answers these questions positively. It proposes novel tighter UBs, a generic framework to evaluate UBs, efficient pruning strategies based on these UBs, and a novel algorithm to efficiently mine HAUIs.

**Contributions.** The major contributions of this paper are summarized as follows. First, we propose *four novel tighter* UBs named $\overline{aub}_1$, $\overline{aub}$, $\overline{iaub}$ and $\overline{laub}$ based on a novel *vertical* representation of UBs (answering $Q_1$). Second, a *generic framework* for evaluating the pruning ability of UBs based on anti-monotone-like criteria is proposed. Based on these UBs, *three pruning strategies* are proposed to eliminate unpromising candidates early, which dramatically reduce the search space (answering $Q_2$). Third, to decrease the execution time, the *diffset* technique [21] is adapted for HAUIM. This allows to quickly compute auxiliary information about the average-utility of itemsets and the proposed UBs. Moreover, a novel *structure* named the *IDUL* prefix tree is used to maintain HAUI candidates and a novel algorithm named dHAUIM is designed for efficiently mining the complete set of HAUIs (answering $Q_3$). Finally, we show empirical evidence that dHAUIM outperforms the state-of-the-art algorithms in terms of runtime and number of joins on both real-life and synthetic databases.

The rest of this paper is organized as follows. Section 2 introduces preliminary concepts and notations related to high utility itemset mining, and defines the problem of HAUIM. Section 3 presents theoretical results including four novel UBs, a generic framework for evaluating UBs and three efficient techniques to prune unpromising candidates early. Section 4 proposes an efficient algorithm named dHAUIM for mining HAUIs. Section 5 presents the experimental evaluation. Finally, Section 6 draws a conclusion and discusses future work.

## 2 Preliminary Concepts and Problem Statement

This section introduces fundamental concepts and notations related to high average-utility itemset mining. They are necessary for explaining the theoretical results proposed in the next sections.

### 2.1 Preliminary concepts

Let $\mathcal{A} \overset{\text{def}}{=} \{a_j, j \in J \overset{\text{def}}{=} \{1, 2, \ldots, M\}\}$ be a finite set of distinct items (e.g. products sold in a retail store). An itemset $A$ is a subset of $\mathcal{A}$ and is called $k$-itemset if it contains $k$ items, i.e. $k = |A|$. Without loss of generality, we can assume that all items in each itemset are sorted in ascending order with respect to the lexicographical order relation $\prec$. Moreover, each item $a$ is associated with a positive number $p(a)$ called its unit profit (or *external utility*), which represents its relative importance to the user in terms of criteria such as unit profit, price, weight or importance. The profit vector $\mathcal{P} \overset{\text{def}}{=} (p(a_j), j \in J)$ consists of the unit profits of all items in $\mathcal{A}$. A *q-item* is a pair $(a, q)$, where $a \in \mathcal{A}$ and $q$ is a non-negative number indicating the purchase quantity (or *internal* utility) of the item $a$. A *transaction* or *q-itemset* $t_i$ is a set of q-items, $t_i \overset{\text{def}}{=} \{(a_j, q_{ij}), j \in J_i\}$, where $J_i \subseteq J$ is an index subset of $J$. A quantitative database (QDB) $\mathcal{D}$ is a finite set of transactions, $\mathcal{D} \overset{\text{def}}{=} \{t_i, i \in I \overset{\text{def}}{=} \{1, 2, \ldots, N\}\}$, where each transaction $t_i$ is associated with a unique identifier $TID$ (*e.g.* $TID = i$). The *utility* of a *q-item* $(a, q)$ is denoted and defined as $u((a, q)) \overset{\text{def}}{=} q * p(a)$. Intuitively, it represents the amount of profit yield by the sale of item $a$ in the transaction where the q-item $(a, q)$ appears.

When mining a quantitative database to find high utility patterns, to avoid repeatedly computing *the utility* $u((a_j, q_{ij})) = q_{ij} * p(a_j)$ of a q-item $(a_j, q_{ij})$ in a transaction $t_i$, the utility of each q-item can be calculated once and stored in the corresponding transaction. The transformed database resulting from applying this process to a quantitative database QDB $\mathcal{D}' \overset{\text{def}}{=} \{t'_i \overset{\text{def}}{=} \{(a_j, q'_{ij}), j \in J_i\}, J_i \subseteq$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2833478, IEEE Transactions on Knowledge and Data Engineering



$J, i \in I\}$ is called an *integrated quantitative matrix* (or briefly *integrated matrix*) $\mathcal{Q}$, where $\mathcal{Q}_{NxM} \triangleq \{q'_{ij}, i \in I, j \in J\}$, and $q'_{ij}$ is defined as follows,

$$q'_{ij} \triangleq \begin{cases} q_{ij} * p(a_j), if\ (a_j, q_{ij}) \in t_i \\ 0 \qquad\qquad , otherwise \end{cases}, \forall i \in I, j \in J.$$

Furthermore, consider an operator $\rho: 2^{\mathcal{A}} \rightarrow 2^{\mathcal{T}}$ defined as follows: $\forall A \subseteq \mathcal{A}: A \neq \emptyset, \rho(A) \triangleq \{t_i \in \mathcal{D} \mid q'_{ij} > 0,\ \forall a_j \in A\}$ and as convention $\rho(\emptyset) \triangleq \mathcal{D}$. Intuitively, this operator maps each non-empty itemset $A$ to transactions where all items in $A$ have purchase quantities greater than zero. Hereafter, we only consider itemsets in the set $\mathcal{IS} \triangleq \{A \subseteq \mathcal{A} \mid \rho(A) \neq \emptyset\}$, that is itemsets appearing in at least one transaction.

**Example 1** (*Running example*). *Table 1 and Table 2 show an example QDB $\mathcal{D}$ containing six transactions and five items $(a, b, c, d, e)$, and a profit vector $\mathcal{P}$ indicating the unit profits of items. The integrated matrix $\mathcal{Q}$ representing the QDB $\mathcal{D}$ and the profit vector $\mathcal{P}$ is shown in Table 3 as running example. For brevity, in the rest of this paper, an itemset $\{b, d\}$ is denoted as $bd$, and a set of transactions $\{t_1, t_3, t_5\}$ is denoted as 135.*

Table 1. A QDB $\mathcal{D}$.

| TID \ Item $a_j$ | $a$ | $b$ | $c$ | $d$ | $e$ |
|---|---|---|---|---|---|
| $t_1$ | 0 | 2 | 10 | 3 | 5 |
| $t_2$ | 1 | 0 | 0 | 2 | 3 |
| $t_3$ | 0 | 5 | 15 | 4 | 10 |
| $t_4$ | 6 | 0 | 5 | 3 | 1 |
| $t_5$ | 5 | 3 | 2 | 2 | 6 |
| $t_6$ | 3 | 4 | 0 | 0 | 6 |

Table 2. A profit vector $\mathcal{P}$.

| $a_j$ | $a$ | $b$ | $c$ | $d$ | $e$ |
|---|---|---|---|---|---|
| $p(a_j)$ | 6 | 80 | 10 | 2 | 1 |

Table 3. The integrated matrix $\mathcal{Q}$ corresponding to Tables 1-2

| TID \ Item | $a$ | $b$ | $c$ | $d$ | $e$ |
|---|---|---|---|---|---|
| $t_1$ | 0 | 160 | 100 | 6 | 5 |
| $t_2$ | 6 | 0 | 0 | 4 | 3 |
| $t_3$ | 0 | 400 | 150 | 8 | 10 |
| $t_4$ | 36 | 0 | 50 | 6 | 1 |
| $t_5$ | 30 | 240 | 20 | 4 | 6 |
| $t_6$ | 18 | 320 | 0 | 0 | 6 |

**Definition 1** (*Utility and average-utility measures* [12]). *Consider any itemset $A$ in $\mathcal{IS}$ and transaction $t_i$ in $\mathcal{D}$. The utility of $A$ in the transaction $t_i$ is denoted and calculated as the sum $u(A, t_i) \triangleq \sum_{j: a_j \in A} q'_{ij}$ according to the $i^{th}$ row of the integrated matrix $\mathcal{Q}$. The utility $u: \mathcal{IS} \rightarrow R_+ \triangleq \{x \in R \mid x > 0\}$ of $A$ (in $\mathcal{D}$) is denoted and defined as:*

$$u(A) \triangleq \sum_{t_i \in \rho(A)} u(A, t_i).$$

*The average-utility (AU) measure $au: \mathcal{IS} \rightarrow R_+$ of each itemset $A \in \mathcal{IS}$ is denoted and defined as:*

$$au(A) \triangleq u(A)/|A|.$$

Intuitively, the utility of an itemset $A$ in a QDB is the amount of profit yield by the sale of the itemset $A$ in transactions where it appears, while the average-utility is the utility divided by the itemset length. Since the utility $u(A, t_i)$ is calculated based on the $i^{th}$ *row* of $\mathcal{Q}$, the value $u(A)$ is said to be represented in *horizontal form*.

For example, $\rho(bd) = 135$, the average-utility of the itemset $bd$ is calculated as $au(bd) = u(bd) / 2 = ((160 + 6) + (400 + 8) + (240 + 4)) / 2 = 409$.

**Definition 2** (*Transaction utility, the total utility of a QDB* [12]). *The transaction utility of a transaction $t_i$ is defined and denoted as $tu(t_i) \triangleq \sum_{q'_{ij} > 0} q'_{ij}$. The total utility of a QDB $\mathcal{D}$ is denoted as $TU$ and defined as the sum of its transaction utilities, that is $TU \triangleq \sum_{t_i \in \mathcal{D}} tu(t_i)$.*

For example, the transaction utility of the transaction $t_1$ is $tu(t_1) = 160 + 100 + 6 + 5 = 271$ and the total utility of $\mathcal{D}$ is $TU = 271 + 13 + 568 + 93 + 300 + 344 = 1589$.

### 2.2 Problem statement

Let $mu$ be a minimum average-utility threshold defined by the user (a positive number). An itemset $A$ in $\mathcal{IS}$ is called high average-utility (HAU) iff[1] $au(A) \geqslant mu$. Otherwise, it is called unpromising or low average utility (LAU). The problem of high average-utility itemset mining is to find the set of all HAUIs, $HAUS(mu)$ or briefly $HAUS \triangleq \{A \in \mathcal{IS} \mid au(A) \geqslant mu\}$. Note that the minimum threshold $mu$ can be also defined as $mu \triangleq \delta * TU$, where $\delta$ is a minimum percentage threshold set by the user.

## 3 Theoretical Results

In this section, we present new theoretical results for the design of an efficient algorithm for mining high average-utility itemsets. In particular, this section explains how the utility measure $u$ and novel upper-bounds on the average-utility measure $au$ can be defined using a vertical form (columns of the $\mathcal{Q}$ matrix), contrarily to previous studies that have used the horizontal form. These results are the basis of the proposed algorithm, presented in Section 4.

Let $v_j(A) \triangleq \sum_{t_i \in \rho(A)} q'_{ij}$ be the column utility of an itemset $A$ in the $j^{th}$ *column* of the $\mathcal{Q}$ matrix. Furthermore, let $\mathcal{V}(A) \triangleq (v_j(A), j \geqslant minInd(A))$ be the set of all column utility values associated to $A$ starting from the $minInd(A)$ *column,* where $minInd(A) \triangleq min\{j \in J \mid a_j \in A\}$ is the minimum index (or first index) of items $a_j$ in $A$. As a convention, $minInd(\emptyset) \triangleq 1$. Based on these definitions, the utility of $A$ is defined in vertical form as follows.

**Lemma 1** (*Vertical form of the utility $u(A)$*). *For each itemset $A \in \mathcal{IS}$, $u(A) = \sum_{j: a_j \in A} v_j(A)$.* (1)

**Proof**. $u(A) \triangleq \sum_{t_i \in \rho(A)} \left[\sum_{a_j \in A} q'_{ij}\right] = \sum_{a_j \in A} \left[\sum_{t_i \in \rho(A)} q'_{ij}\right] = \sum_{a_j \in A} v_j(A)$. □

Since $v_j(A)$ is calculated based on the $j^{th}$ *column* of $\mathcal{Q}$, the utility $u(A)$ in equation (1) is said to be represented using the *vertical form*. For example, consider that $A = bd$. Then, $minInd(A) = 2$ (the index of $b \equiv a_2$), $v_1(A) = q'_{11} + q'_{31} + q'_{51} = 30$, similarly $\mathcal{V}(A) = (30, 800, 270, 18, 21)$, and $u(A) = v_2(A) + v_4(A) = 818$.

[1] iff means "if and only if"

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2833478, IEEE Transactions on Knowledge and Data Engineering



During the expansion of the search space of HAU candidate itemsets, an itemset $P$ is extended with an itemset $B$ such that $P \prec B$ (*i.e.* $\forall a \in P, \forall b \in B, a \prec b$). The result is an extension $C$ (also called a forward extension) of $P$ (with itemset $B$), denoted as $C = P \oplus B$, *i.e.* $C = P \cup B$ under the condition that $P \prec B$. In this situation, the itemset $P$ is said to be a *prefix* of $C$. For example, if $B = \{b\}$ and $P \prec \{b\}$, an item-extension of $P$ (with item $b$) is $P \oplus \{b\}$, concisely denoted as $Pb$.

Note that the $au$ measure is *neither monotonic nor anti-monotonic*. *Indeed*, for example, $a \subset ab \subset abc$, but $au(a) = 90 < au(abc) = 290/3 < au(ab) = 304$. To design pruning strategies for reducing the search space in HAUIM, an UB named $auub$ has been proposed [14]. The $auub$ measure is an *upper bound* on $au$, which satisfies the *DCP* (or *anti-monotonicity* property), i.e. $\forall A \subseteq B \subseteq \mathcal{A}$, it follows that $au(A) \leqslant auub(A)$ and $auub(B) \leqslant auub(A)$. Therefore, if $A$ is low-$auub$, i.e. $auub(A) < mu$, then all its super itemsets $B$ are also low-$auub$ and LAU. To prune more unpromising candidates from the search space, several upper-bounds on the $au$ measure have been proposed [16], [18], [19]. However, the set of candidate itemsets considered when using these upper-bounds is often very large. To overcome this problem, this article proposes four new UBs that are tighter than $auub$, named $\overline{aub}_1$, $\overline{aub}$, $\overline{iaub}$ and $\overline{laub}$, which can be quickly calculated using the proposed *vertical form* $v_j(A)$.

The key idea behind the four new UBs proposed in this article is the following. For any matrix $Q_{NxM}$ and two non-empty index subsets $I' \subseteq I$, $J' \subseteq J$, the following inequation holds:

$$max\{\sum_{i \in I'} q_{ij}, j \in J'\} \leqslant \sum_{i \in I'} max\{q_{ij}, j \in J'\}.$$

In this inequation, each term $max\{q_{ij}, j \in J'\}$ is calculated in *horizontal* form, and has been used in previous UBs such as $auub$ and $rtub$, $lub$ [18], while the term $v_j \triangleq \sum_{i \in I'} q_{ij}$ is calculated in *vertical* form, and is used in the novel UBs presented in this paper. Based on this inequation, it can be found that UBs based on the *vertical form* are often better than those based on the *horizontal* form. The next subsection presents the four novel upper-bounds. Then, the following subsections introduce a framework to evaluate and compare upper-bounds for HAUIM, and novel pruning strategies.

### 3.1 Four novel UBs

In this subsection, to answer the question ($Q_1$), we propose four novel UBs named $\overline{aub}_1$, $\overline{aub}$, $\overline{iaub}$ and $\overline{laub}$ that are gradually tighter than the traditional UB $auub$ on $au$, have gradually weaker *anti-monotone-like* properties and pruning effects as shown below. Firstly, we define two new UBs gradually tighter than $auub$ named $\overline{aub}_1$ and $\overline{aub}$, which satisfy two stronger *anti-monotone-like* properties, and have respectively *strong width* and *width pruning* effects as it will be explained in next sections.

**Definition 3** (*The UB* $auub$ [16] *and two novel UBs, named* $\overline{aub}_1$ *and* $\overline{aub}$). *For each itemset* $A \in \mathcal{IS}$, *the traditional UB of* $A$ *is denoted and defined as* $auub(A) \triangleq \sum_{t_i \in \rho(A)} max\{q'_{ij}, j \in J\}$. *Two novel UBs of* $A$, *named* $\overline{aub}_1$ *and* $\overline{aub}$, *are defined as:*

$$\overline{aub}_1(A) \triangleq max\{v_k(A) \mid k \geqslant 1\}, \quad (2)$$

$$\overline{aub}(A) \triangleq \overline{aub}_{minInd(A)}(A), \quad (3)$$

*where* $\overline{aub}_j(A) \triangleq max\{v_k(A) \mid k \geqslant j\}, \forall j \in J$.

For example, consider the itemset $A = cd$. It is found that $minInd(A) = 3$, $\rho(A) = 1345$, $auub(A) = 160 + 400 + 50 + 240 = 850$. Moreover, $v_1(A) = q'_{11} + q'_{31} + q'_{41} + q'_{51} = 0 + 0 + 36 + 30 = 66$ and similarly $\mathcal{V}(A) = (66, 800, 320, 24, 22)$. The average-utility of $A$ is $au(A) = (v_3(A) + v_4(A)) / 2 = 172$, $\overline{aub}_1(A) = max\{66, 800, 320, 24, 22\} = 800$, $\overline{aub}(A) = \overline{aub}_3(A) = max\{320, 24, 22\} = 320$. It is observed that $au(A) < \overline{aub}(A) < \overline{aub}_1(A) < auub(A)$, i.e. that the two proposed upper-bounds are tighter than $auub$.

To design an upper-bound that is tighter than $\overline{aub}(A)$, the following observation is made. Consider that $A = acd$, which does not contain the item $a_2 \equiv b$, and that $minInd(A) = 1$. The value $v_2(A)$ in the definition $\overline{aub}(A) = \overline{aub}_1(A) \triangleq max\{v_k(A) \mid k \geqslant 1\}$ may result in a value $\overline{aub}(A)$ that is larger than $max\{v_k(A) \mid k = 1, 3, 4 \text{ or } k > 4\}$. This observation leads to an improved UB $\overline{iaub}$ on $au$ that is tighter than $\overline{aub}$ and another UB $\overline{laub}$ that is incomparable with $\overline{iaub}$. Both of them will satisfy *weaker anti-monotone-like* properties and have the *depth pruning* effect as shown in Proposition 1 and Theorem 1 below.

**Definition 4** (*Two novel UBs, named* $\overline{iaub}$, $\overline{laub}$). *For each itemset* $A = Pa_m \in \mathcal{IS}$, *a third improved UB of* $A$ *is denoted and defined as follows*:

$$\overline{iaub}(A) \triangleq max\{v_k(A) | a_k \in A \text{ or } k > m\}. \quad (4)$$

*A fourth looser UB named* $\overline{laub}$ *of* $A$ *is defined as*

$$\overline{laub}(A) \triangleq au(A) + \overline{aub}_{m+1}(A). \quad (5)$$

For example, $\overline{iaub}(a) = \overline{aub}_1(a) = max\{v_i(a), i = 1, \ldots, 5\} = max\{90, 560, 70, 14, 16\} = 560$ and $\overline{iaub}(d) = \overline{aub}_4(d) = max\{v_4(d), v_5(d)\} = 28$. Consider the itemset $A = acd$. We have $\rho(A) = 45$, $minInd(A) = 1$, $d = a_4$, $m = 4$, $auub(A) = max\{36, 50, 6, 1\} + max\{30, 240, 20, 4, 6\} = 290$, $au(A) = (92 + 54) / 3 = 146/3$. Moreover, $v_1(A) = q'_{41} + q'_{51} = 36 + 30 = 66$ and similarly, $\mathcal{V}(A) = (66, 240, 70, 10, 7)$. Thus, $\overline{aub}_1(A) = \overline{aub}(A) = max\{v_i(A), i \geqslant 1\} = 240$, $\overline{iaub}(A) = max\{v_k(A) \mid k = 1, 3, 4 \text{ or } k > 4\} = 70$. Moreover, $\overline{laub}(A) = au(A) + max\{v_5(A)\} = 167/3$. Thus, in this example, $au(A) < \overline{laub}(A) < \overline{iaub}(A) < \overline{aub}(A) \leqslant \overline{aub}_1(A) < auub(A)$.

In the following, for any UB $ub$, an itemset $A$ is called high-$ub$ iff $ub(A) \geqslant mu$. Let us denote $HAUB_1 \triangleq \{A \in \mathcal{IS} \mid \overline{aub}_1(A) \geqslant mu\}$, $HAUB \triangleq \{A \in \mathcal{IS} \mid \overline{aub}(A) \geqslant mu\}$ and $HIAUB \triangleq \{A \in \mathcal{IS} \mid \overline{iaub}(A) \geqslant mu\}$ as the sets of all high-$\overline{aub}_1$, high-$\overline{aub}$ and high-$\overline{iaub}$ itemsets respectively.

Recently, several UBs have been proposed such as $rtub$, $lub$ [18], $lubau$, $tubau$ [19] and $mau$ [20]. Moreover, this article has proposed four novel UBs. Thus, a problem is how to evaluate and compare all these UBs. A simple way is to compare their values. Let $ub_1$ and $ub_2$ be two UBs. $ub_1$ is said to be *tighter* than $ub_2$, denoted as $ub_1 \ll ub_2$, iff $ub_1(A) \leqslant ub_2(A)$, $\forall A \in \mathcal{IS}$. However, two upper-bounds may be *incomparable*, *i.e.* if there exist two different itemsets $A$ and $B$ such that $ub_1(A) > ub_2(A)$ and $ub_1(B) < ub_2(B)$. A second way of comparing upper-bounds is to consider whether they respect the anti-monotonicity property ($\mathcal{AM}$) or variations of that property. The $auub$ upper-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2833478, IEEE Transactions on Knowledge and Data Engineering



bound satisfies $\mathcal{AM}$, which allows pruning many low-*au* itemsets. However, $auub$ provides a too loose overestimation of the average-utility of itemsets, which can result in large candidate sets, especially for low $mu$ values. To prune more candidates, several UBs (including those proposed in this paper), respect variations of the $\mathcal{AM}$. To compare upper-bounds and answer the second research question **($Q_2$)** of this study, the next subsection proposes a generic framework for evaluating UBs using *anti-monotone-like* ($\mathcal{AML}$) criteria. Based on this framework, the following subsection develops different pruning strategies and evaluates UBs in terms of their pruning ability.

### 3.2 Generic framework for evaluating UBs

**Definition 5** (*Anti-monotone-like* ($\mathcal{AML}$) *criteria of UBs*). *Let $C, D, E, P, R$ be itemsets in $\mathcal{IS}$ and $ub$ be an UB on the average utility $au$, i.e. $au(P) \leqslant ub(P), \forall P \in \mathcal{IS}$. Then $ub$ is said to satisfy*

*a. the anti-monotonicity property (or criterion), denoted as $\mathcal{AM}(ub)$, iff $ub(C) \leqslant ub(P), \forall C \supseteq P$.*

*b. the anti-monotonicity property w.r.t. bi-directional extension, denoted as $\mathcal{BIDEAM}(ub)$, iff $ub(C) \leqslant ub(P)$, $\forall P = R \oplus D, C = R \oplus E$ such that $R \neq \emptyset$ and $D \subseteq E$.*

*c. the anti-monotonicity property w.r.t. forward extension, denoted as $\mathcal{FEAM}(ub)$, iff $ub(C) \leqslant ub(P)$, $\forall P, C = P \oplus E$ such that $P \neq \emptyset$ (i.e. $C$ is a forward extension of the non-empty prefix $P$).*

*d. the depth pruning condition by forward extension, denoted as $\mathcal{DPC}(ub)$, if $au(C) \leqslant ub(P)$, $\forall P, C = P \oplus E$ such that $P \neq \emptyset$.*

It is clear that if $\mathcal{AM}(ub)$, then $ub$ satisfies the downward closure property. This means that if $P$ is *low-ub*, i.e. $ub(P) < mu$, then all its super itemsets $C$ are also *low-ub* (because $ub(C) \leqslant ub(P) < mu$ ) and thus $C$ can be pruned since it is also *low-au*, i.e. $au(C) \leqslant ub(C) < mu$.

**Proposition 1** (*Relation between the $\mathcal{AML}$ criteria of UBs*). *Let $ub$ be an UB. Then,*
$\mathcal{AM}(ub) \Rightarrow \mathcal{BIDEAM}(ub) \Rightarrow \mathcal{FEAM}(ub) \Rightarrow \mathcal{DPC}(ub)$.

**Proof**. The correctness of the assertion "$\mathcal{AM}(ub) \Rightarrow \mathcal{BIDEAM}(ub)$" is obvious. The one "$\mathcal{BIDEAM}(ub) \Rightarrow \mathcal{FEAM}(ub)$" is implied by considering $D = \emptyset$. Finally, because $au(C) \leqslant ub(C) \leqslant ub(P), \forall P \subseteq C = P \oplus E: P \neq \emptyset$, it follows that "$\mathcal{FEAM}(ub) \Rightarrow \mathcal{DPC}(ub)$". □

In this context, we can say that, $\mathcal{AM}$ (or $\mathcal{DPC}$) is the strongest (or weakest respectively) $\mathcal{AML}$ *criterion* among the four above ones.

**Lemma 2** (*Properties of $\overline{aub_j}$*).

*a. (Decreasing property of $\overline{aub_j}(A)$ w.r.t. $j \in J$) For each itemset $A \in \mathcal{IS}$, $\forall k \geqslant j \geqslant 1$, $\overline{aub_k}(A) \leqslant \overline{aub_j}(A) \leqslant \overline{aub_1}(A)$.*

*b. (Anti-monotonicity property of $\overline{aub_j}(A)$ w.r.t. $A \in \mathcal{IS}$) For each index $j \in J$, then $\mathcal{AM}(\overline{aub_j})$.*

**Proof**.

a. Since $\{v_i(A), \forall i \geqslant k\} \subseteq \{v_i(A), \forall i \geqslant j\} \subseteq \{v_i(A), \forall i \geqslant 1\}$, $\forall k \geqslant j \geqslant 1$ , it follows that $\overline{aub_k}(A) \leqslant \overline{aub_j}(A) \leqslant \overline{aub_1}(A)$.

b. Since $\forall A \subseteq B \in \mathcal{IS}$, $\rho(A) \supseteq \rho(B)$, then $\forall k \geqslant j$, $v_k(B) = \sum_{t_i \in \rho(B)} q'_{ik} \leqslant \sum_{t_i \in \rho(A)} q'_{ik} = v_k(A) \leqslant max\{v_k(A), \forall k \geqslant j\} = \overline{aub_j}(A)$ , it follows that $max\{v_k(B), \forall k \geqslant j\} = \overline{aub_j}(B) \leqslant \overline{aub_j}(A)$. □

**Theorem 1** (*Properties of $\overline{iaub}$, $\overline{aub}$, $\overline{aub}_1$ and $auub$*).

*a. (Anti-monontonicity property of $\overline{aub}_1$ and $auub$) $\mathcal{AM}(\overline{aub}_1)$ and $\mathcal{AM}(auub)$.*

*b. (Anti-monontonicity property of $\overline{aub}$ w.r.t. bi-directional extension) $\mathcal{BIDEAM}(\overline{aub})$.*

*c. (Anti-monontonicity property of $\overline{iaub}$ w.r.t. forward extension) $\mathcal{FEAM}(\overline{iaub})$.*

*d. (Depth pruning condition by forward extension of $\overline{laub}$) $\mathcal{DPC}(\overline{laub})$.*

*e. $auub$ , $\overline{aub}_1$ , $\overline{aub}$ and $\overline{iaub}$ are respectively gradually tighter upper-bounds on $au$, i.e. $au(A) \leqslant \overline{iaub}(A) \leqslant \overline{aub}(A) \leqslant \overline{aub}_1(A) \leqslant auub(A), \forall A \in \mathcal{IS}$.*
*Thus, $HAUS \subseteq HIAUB \subseteq HAUB \subseteq HAUB_1 \subseteq \{A \in \mathcal{IS} \mid auub(A) \geqslant mu\}$.*

*f. $\overline{laub}$ is also an UB on $au$; $\overline{laub}$ and $\overline{iaub}$ are incomparable.*

**Proof**. Given any itemsets $C, D, E, P, R \in \mathcal{IS}$ such that $P \neq \emptyset$ and $P \subseteq C$, we always have $\rho(C) \subseteq \rho(P)$ and $v_k(C) \leqslant v_k(P), \forall k \in J$.

a. The first assertion is implied from Lemma 2.b.

b. For $P = R \oplus D$ and $C = R \oplus E$ such that $R \neq \emptyset$ and $D \subseteq E$ , then $P \subseteq C$ , $minInd(R) = minInd(P) = minInd(C)$, $v_k(C) \leqslant v_k(P)$, $\forall k \geqslant minInd(R)$. Thus, $\overline{aub}(C) \leqslant \overline{aub}(P)$.

c. Consider $C = P \oplus E$ with non-empty $P$ and $E$. Then $maxInd(P) < minInd(E)$, so $\{k \in J \mid a_k \in P \oplus E \text{ or } k > maxInd(E)\} \subseteq \{k \in J \mid a_k \in P \text{ or } k > maxInd(P)\}$ . Thus, $\overline{iaub}(C) \leqslant \overline{iaub}(P)$.

d. $\forall P \subseteq C = P \oplus E$ such that $P \neq \emptyset$ , set $\Delta \stackrel{\text{def}}{=} max\{v_j(P) \mid j > maxInd(P)\}$, then $\overline{laub}(P) - au(C) = \Delta + \frac{u(P)}{|P|} - \frac{u(C)}{|P|+|E|} = \Delta + \frac{|P|.(u(P)-u(C))+|E|.u(P)}{|P|.(|P|+|E|)} \geqslant \Delta - \frac{u(C)-u(P)}{|P|+|E|} \geqslant 0$, so $au(C) \leqslant \overline{laub}(P)$, because

$$\begin{aligned} u(C) - u(P) &= \sum_{t_i \in \rho(C)} \left[\sum_{a_j \in C} q'_{ij}\right] - \sum_{t_i \in \rho(P)} \left[\sum_{a_j \in P} q'_{ij}\right] \\ &\leqslant \sum_{t_i \in \rho(P)} \left[\sum_{a_j \in C} q'_{ij} - \sum_{a_j \in P} q'_{ij}\right] \\ &= \sum_{t_i \in \rho(P)} \left[\sum_{a_j \in E} q'_{ij}\right] = \sum_{a_j \in E} \left[\sum_{t_i \in \rho(P)} q'_{ij}\right] \\ &= \sum_{a_j \in E} v_j(P) \leqslant |E| .\Delta \leqslant (|P| + |E|).\Delta. \end{aligned}$$

e. It is clear that $\overline{iaub}(C) \geqslant max\{v_k(C) \mid a_k \in C\} \geqslant \sum_{a_k \in C} v_k(C) / |C| = au(C)$.
Moreover, $auub(C) \geqslant \sum_{t_i \in \rho(C)} q'_{ij} = v_j(C)$, $\forall j \geqslant 1$. Hence, $auub(C) \geqslant max\{v_j(C) \mid j \geqslant 1\} = \overline{aub}_1(C) \geqslant \overline{aub}_{minInd(C)}(C) = \overline{aub}(C)$ by Lemma 2.a. Thus, we only need to prove that $\overline{aub}(C) \geqslant \overline{iaub}(C)$. Since $\{k \in J \mid k \geqslant minInd(C)\} \supseteq \{k \in J \mid a_k \in C \text{ or } k > maxInd(C)\}$, then $\overline{aub}(C) \geqslant \overline{iaub}(C)$ . Hence, $auub(C) \geqslant \overline{aub}_1(C) \geqslant \overline{aub}(C) \geqslant \overline{iaub}(C) \geqslant au(C)$.

f. By the definition of $\overline{laub}$, $au(C) \leqslant \overline{laub}(C)$. The remaining assertion is shown below. □

**Example 2** (*$\overline{laub}$ and $\overline{iaub}$ are incomparable*). *Consider two itemsets $A = b$ and $B = bd$ , then $\overline{laub}(A) = 1390 > auub(A) = \overline{aub}_1(A) = \overline{aub}(A) = \overline{iaub}(A) = au(A) = 1120$ and $\overline{iaub}(B) = 800 > \overline{laub}(B) = 430 > au(B) = 409$.*

Thus, $\overline{laub}$ (satisfying only $\mathcal{DPC}$, the weakest $\mathcal{AML}$ *criterion* by Proposition 1) and $\overline{iaub}$ (satisfying $\mathcal{FEAM}$) are

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2833478, IEEE Transactions on Knowledge and Data Engineering



*incomparable*. However, the value of $\overline{laub}$ may be larger than the largest $auub$, $\overline{aub}_1$ and $\overline{aub}$, whereas $\overline{iaub}$ is always tighter than $\overline{aub}$. Thus, the amplitude value (the difference between the minimum and maximum values) of $\overline{laub}$ is often greater than that of $\overline{iaub}$.

Having studied properties of the four proposed upper-bounds and their relationship to previous upper-bounds, the next important problem to solve is how to apply the four proposed UBs for designing efficient pruning strategies to answer the second question ($Q_2$).

### 3.3 Three novel pruning strategies based on the proposed UBs

Although $\overline{aub}_1$, $\overline{aub}$ and $\overline{iaub}$ are respectively gradually tighter UBs on $au$, and $HAUS \subseteq HIAUB \subseteq HAUB \subseteq HAUB_1$, each UB has its own pruning ability or effect. For any UB $ub$, we say that the *pruning condition* $PC_{ub}(P)$ for a non-empty itemset $P$ holds iff $ub(P) < mu$ (or $P$ is low-*ub*). For the sake of brevity, we denote the set of $P$ and all itemset extensions of $P$ as $branch(P)$. Furthermore, the notation $[P] \stackrel{\text{def}}{=} \{Px, \ldots, Py, \ldots, Pz, \ldots\}$ will denote the equivalence class consisting of all item-extensions of $P$ (itemsets having the same prefix $P$). Based on these concepts, the search space of HAUIM can be viewed as a *prefix-tree*, where each node represents an itemset, such that the root is the empty set and each child of a node is single item-extension of that itemset. These definitions will be used in the proposed dHAUIM algorithm, presented in Section 4. Based on Theorem 1, the three proposed pruning strategies are described next.

**Pruning Strategy 1** (*Depth Pruning (DP) strategy w.r.t. forward extensions based on* $\overline{iaub}$ *and* $\overline{laub}$). For any non-empty itemset $P$, if $PC_{\overline{iaub}}(P)$ or $PC_{\overline{laub}}(P)$ hold, the whole $branch(P)$ of the search space can be pruned.

Since $\overline{iaub}$ and $\overline{laub}$ are incomparable, both of these UBs should be used to eliminate unpromising candidate branches of the prefix-tree.

**Pruning Strategy 2** (*Width Pruning (WP) strategy w.r.t. bi-directional extensions on projected databases based on* $\overline{aub}$). For the second UB $\overline{aub}$, which is *larger* than $\overline{iaub}$, if $PC_{\overline{aub}}(Ry)$ holds for a *non-empty* itemset $R$ and item-extension $Ry$ in $[R]$, the itemset $Ry$ can be removed from the set $[R]$, i.e. not only the whole $branch(Ry)$ is immediately pruned, but also all branches $branch(Rxy)$ and $branch(Ryz)$ are eliminated from the search tree (where $Rxy$ and $Ryz$ are respectively the backward and forward extensions of $Ry$). In other words, such item $y$ will not appear in $branch(R)$ and we can remove $y$ from the projected database [16] of $R$.

**Pruning Strategy 3** (*Strong Width Pruning (SWP) strategy on the initial QDB* $\mathcal{D}$ *based on* $\overline{aub}_1$). For the $\overline{aub}_1$ UB, which is the largest among the four new UBs, if $PC_{\overline{aub}_1}(P)$ holds, then $PC_{\overline{aub}_1}(C)$ also holds for all extensions $C$ of $P$. In particular, for each item $a_j$ of $\mathcal{A}$, if $PC_{\overline{aub}_1}(a_j)$ holds, *i.e.* $\overline{aub}_1(a_j) < mu$, and we can remove $a_j$ from the database $\mathcal{D}$ or delete the $j^{th}$ column (according to $a_j$) of the integrated matrix $\mathcal{Q}$.

Note that, although $\overline{aub}_1$ and $auub$ have the same SWP pruning ability, $\overline{aub}_1$ is tighter than $auub$ by Theorem 1.e. Thus, in terms of value and search space pruning ability, $\overline{aub}_1$ is said to be absolutely better than $auub$.

Obviously, SWP and WP are respectively stronger than WP and DP. Although both $\overline{aub}_1$ and $\overline{aub}$ have the WP ability, $\overline{aub}$ is only used on projected databases for *non-empty* prefixes while $\overline{aub}_1$ can be additionally applied on *the largest initial QDB* $\mathcal{D}$ (according to the *empty* prefix). Generally, the pruning condition $PC_{ub}$ of a looser UB $ub$ can be applied less frequently than a tight upper-bound due to its high overestimation, but it will eliminate much more unpromising candidates if its pruning condition holds. Since the pruning ability of the four proposed UBs are different, $\overline{laub}$ and $\overline{iaub}$ are *incomparable*, the four novel upper-bounds should all be used together to more efficiently reduce the search space in HAUIM. Moreover, one should not be replaced with the others (see Remarks about this in Section 5).

**Example 3** (*Pruning ability of the three strategies*). *The pruning effect of strategies 1-3 is illustrated in Fig. 1 for the running example.*

For $mu = \mu = 900$, Fig. 1.a illustrates the SWP and DP pruning effects of $\overline{aub}_1$ and $\overline{laub}$. Since $\overline{aub}_1(A) < mu$, for all $A \in \{a, c, d\}$, by the SWP strategy 3, we can remove all items $a$, $c$ and $d$ from the QDB $\mathcal{D}$ (thus, it is unnecessary to perform *backward* extensions of $c$ and $d$ with $b$ to obtain $bc$ and $bd$). Moreover, because $\overline{laub}(be) < mu$ and $\overline{laub}(e) < mu$, the whole $branch(e)$ and $branch(be)$ of the search tree are pruned by DP strategy 1. Finally, it is found that there is only one HAU item $b$ with $au(b) = 1120 \geqslant mu$, $HAUS = \{b\}$.

Fig. 1.b illustrates the WP and DP effects of $\overline{aub}$, $\overline{iaub}$ and $\overline{laub}$. For $mu = \mu' = 250$, $R = a$, $D \in \{c, d\}$, we have $[R] = \{ab, ac, ad, ae\}$ and $\overline{aub}(A) < mu$ for $A = R \oplus D \in \{ac, ad\}$ which have the same *non-empty* prefix $a$. Thus, we can apply the WP strategy for such $A$, i.e. not only all *forward* extensions of $A$ (composed of $branch(acd)$, $branch(ace)$ and $branch(ade)$) in $branch(A)$ can be *deeply* pruned, but also all its *backward* extensions (including $branch(abc)$ and $branch(abd)$). In other words, $ac$ and $ad$ can be eliminated from $[R]$, that is $[R] = \{ab, ae\}$, or equivalently $c$ and $d$ can be removed from the projected database of $a$. However, for items $b$ and $d$, which have the same *empty* prefix, $\overline{iaub}(d) = \overline{aub}(d) = 28 < mu$. Hence, for the item $d$, only the $branch(d)$ can be *deeply pruned* (removed by DP), but we cannot apply the WP and SWP strategies because $\overline{aub}(bd) = \overline{iaub}(bd) = 800 > au(bd) = 409 > mu$. Thus, $bd \in HAUS$, i.e. its *backward* extension $bd$ cannot be pruned and we cannot delete $d$ from $\mathcal{D}$. Thus, the WP effect of the tighter $\overline{aub}$ UB is strictly weaker than the SWP effect of the larger $\overline{aub}_1$ UB. Furthermore, consider the DP effect of $\overline{iaub}$ for $mu = \mu'' = 75$. Since $\overline{iaub}(A) < mu$ but $\overline{aub}(A) \geqslant mu$ for all $A \in \{ac, ad\}$, only *forward* extensions in $branch(A)$ can be *deeply* pruned. The *backward* extension $branch(B)$ of $A$ where $B \in \{abc, abd\}$ cannot be pruned because $au(B) \geqslant mu$. Since $\overline{laub}(ac) > mu > \overline{iaub}(ac)$, the $branch(ac)$ can be pruned by $\overline{iaub}$, but not by $\overline{laub}$. On the other hand, for $mu = \mu''' = 55$, the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2833478, IEEE Transactions on Knowledge and Data Engineering



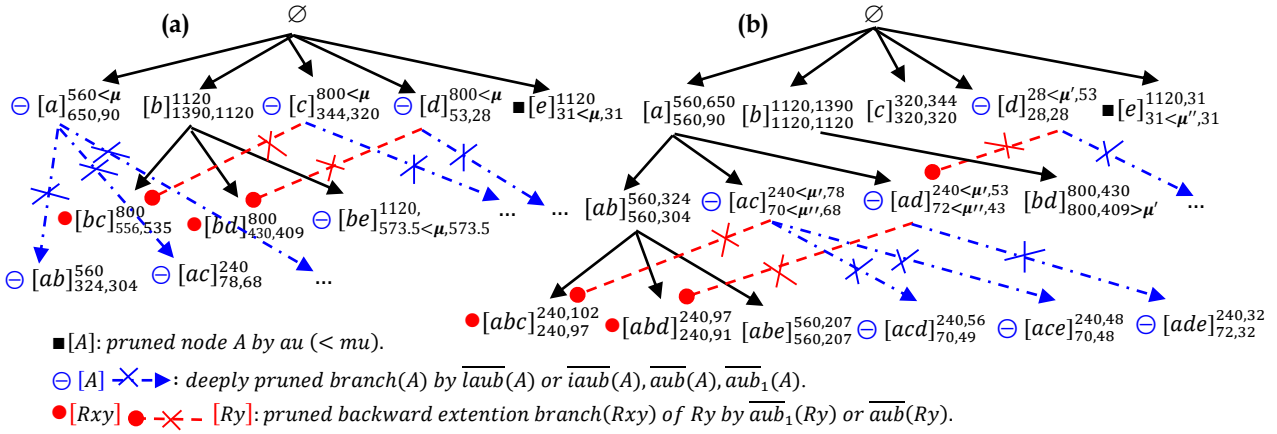


**Fig. 1**. Illustration of pruning strategies - (a) each node $[A]^{\overline{aub_1}(A)}_{\overline{laub}(A),au(A)}$ is considered for pruning with strategies 3 and 1 for $mu = \mu = 900$; (b) each node $[A]^{\overline{aub}(A),\overline{laub}(A)}_{\overline{iaub}(A),au(A)}$ is considered for pruning with strategy 2 for $mu = \mu' = 250$ and strategy 1 with $mu = \mu'' = 75$.

$branch(ad)$ is *deeply* pruned by $\overline{laub}$ but not by $\overline{iaub}$, because $\overline{laub}(ad) < mu < \overline{iaub}(ad)$. In other words, $\overline{laub}$ and $\overline{iaub}$ are incomparable, thus both $\overline{laub}$ and $\overline{iaub}$ should be used to more efficiently prune branches of the prefix search tree.

From a theoretical perspective, it is also relevant to compare the four proposed UBs with the previous UBs, *auub* [16], *rtub* and *lub* [18], *lubau* and *tubau* [19], and *mau* [20].

**Definition 6** (*lubau, tubau* [19] *and lub* [18] *UBs*). *Let there be an itemset* $A \in \mathcal{IS}$. *Assume that items* $\{a_j, j \in J\}$ *are sorted in descending order w.r.t.* $u(a_j)$. *Then:*

*a. the lower-upper-bound average-utility of A is denoted and defined as* $lubau(A) \stackrel{def}{=} \sum_{a_j \in A} u(a_j) \,/\, |A|$;

*b. the tighter-upper-bound average-utility of A is denoted and defined as* $tubau(A) \stackrel{def}{=} [u(B) + u(a_{i_{k+1}})] \,/\, 2$, *where* $B = \{a_{i_1}, a_{i_2}, \dots, a_{i_k}\}$ *and* $A = B \cup \{a_{i_{k+1}}\}$.

*c. the looser upper-bound of A is denoted and defined as* $lub(A) \stackrel{def}{=} au(A) + remu(A)$, *where items* $\{a_j, j \in J\}$ *are assumed to be sorted in ascending order w.r.t.* $auub(a_j)$ *and* $remu(A) \stackrel{def}{=} \sum_{t_i \in \rho(A)} max\{q'_{ij}, j > maxInd(A)\}$.

**Definition 7** (*mau measure* [20]). *Let* $A = Pa_m$ *be an itemset in* $\mathcal{IS}$. *Assume that items* $\{a_j, j \in J\}$ *are sorted in ascending order w.r.t.* $auub(a_j)$. *Then, the maximum item utility of remaining items according to A is defined and denoted as* $mau(A) \stackrel{def}{=} \sum_{t_i \in \rho(A)} mau(A, t_i)$, *where*

$$mau(A,t_i) \stackrel{def}{=} \begin{cases} \dfrac{u(A,t_i) + mu(A,t_i) \times mn(A)}{|A| + mn(A)}, & if\ mu(A,t_i) > au(A,t_i) \\ \dfrac{u(A,t_i) + mu(A,t_i)}{|A|+1}, & if\ 0 < mu(A,t_i) \leqslant au(A,t_i) \\ 0, & if\ mu(A,t_i) = 0 \end{cases}$$

$mu(A,t_i) \stackrel{def}{=} max\{u(a,t_i) | \ a \in t_i, a \succ a_m\}$ , $n(a_m,t_i) \stackrel{def}{=} |\{a_k \in t_i | \ k > m\}|$ *and* $mn(A) \stackrel{def}{=} max\{n(a_m,t_i) \mid t_i \in \rho(A)\}$.

**Discussion about the pruning effects of the previous *auub*, *lubau*, *tubau*, *rtub*, *lub* and *mau* UBs.**

(*i*). The new UB $\overline{laub}$ is *tighter* than *lub*. *Indeed*, since for any $k > m$, $v_k(A) = \sum_{t_i \in \rho(A)} q'_{ik} \leqslant \sum_{t_i \in \rho(A)} max\{q'_{ik}, k > m\} = remu(A)$, it follows that $max\{v_k(A) \mid k > m\} \leqslant remu(A)$, and thus $\overline{laub}(A) \leqslant lub(A)$.

(*ii*). The proposed $\overline{aub_1}$, $\overline{aub}$ and $\overline{iaub}$ UBs are *tighter* than *auub* (by Theorem 1.e). However, the *lubau*, *tubau*, *lub* and $\overline{laub}$ UBs are *not tighter* than *auub*, as they may produce values that are greater than *auub* for some itemset, i.e. $\exists A: auub(A) < ub(A)$. Thus, the *lubau*, *tubau*, $\overline{laub}$ and *lub* UBs are said to be *less stable* than the three UBs $\overline{aub_1}$, $\overline{aub}$ and $\overline{iaub}$ because their amplitude can vary more widely. Moreover, *lubau*, *tubau* and *lub* are incomparable; *lubau*, *tubau* and $\overline{laub}$ are also incomparable.

This is illustrated with an example. Consider the integrated QDB $\mathcal{D}' = \{t_1 = ((a, 100)(c, 2)(d, 6)(e, 400)(f, 200)(h, 5), t_2 = ((a, 1)(b, 40)(c, 3)(d, 50)(f, 50)(g, 10)(h, 4)\}$ and the itemsets $A = ac$, $C = abc$, and $D = acd$. It is found that $au(A) = 53 = tubau(A) = lubau(A) < \overline{iaub}(A) = 400 \leqslant \overline{aub}(A) = \overline{aub_1}(A) = 400 < auub(A) = 450 < \overline{laub}(A) = 453 < lub(A) = 503$. Moreover, $tubau(C) = [(41 + 5)] \,/\, 2 = 23 < lubau(C) = (101 + 40 + 5) \,/\, 3 = 146/3$, but $tubau(D) = [(106 + 56)] \,/\, 2 = 81 > lubau(D) = (101 + 5 + 56) \,/\, 3 = 54$. Consider another integrated QDB $\mathcal{D}' = \{t_1 = \{(a_1, 6), (a_2, 3)\}, t_2 = \{(a_1, 11)\}\}$ and the itemset $A = \{a_1, a_2\}$. It is found that $\rho(A) = \{t_1\}$, and $\overline{laub}(A) = lub(A) = 4.5 < auub(A) = 6 < tubau(A) = lubau(A) = [u(a_1) + u(a_2)] \,/\, 2 = 10$.

(*iii*). As stated in [18], [19], the three *lubau*, *tubau* and *lub* UBs satisfy the pruning condition $\mathcal{DPC}$, the weakest $\mathcal{AML}$ criterion, under the following condition: all items $a_j$ must be sorted in descending order *w.r.t.* the utility $u(a_j)$ for *lubau* and *tubau*, and in ascending order *w.r.t.* the $auub(a_j)$ for *lub*. Thus, *lubau*, *tubau* and *lub* are sensitive to the sorting order of items. This leads to difficulties when they need to be used simultaneously to prune unpromising itemsets.

(*iv*). Moreover, note that *lubau* does not satisfy the $\mathcal{BIDEAM}$ and $\mathcal{AM}$ properties, and that the *tubau*, *lub* and $\overline{laub}$ UBs do not satisfy the stronger $\mathcal{AM}$, $\mathcal{BIDEAM}$ and $\mathcal{FEAM}$ properties. These UBs can produce high overestimation of the average utility and as a result, the width pruning strategies 2-3 cannot be used for these UBs.

(*v*). In the projected database of each itemset $R$ (PDB($R$)) [16], if $\exists y = a_i \succ R: auub(Ry) < mu$, then we can remove

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2833478, IEEE Transactions on Knowledge and Data Engineering



$y$ from PDB($R$) and update $auub$ to obtain the tighter UB $rtub$ [18] as follows. For any itemset $C = R \oplus E$, then $rtub(C) \triangleq \sum_{t_k \in \rho(C)} max\{q'_{kj} \mid j \geqslant 1 \text{ and } j \neq i\} \leqslant \sum_{t_k \in \rho(C)} max\{q'_{kj} \mid j \geqslant 1\} \triangleq auub(C)$. Since $auub$ and $rtub$ are created using the horizontal form, this update requires performing additional PDB or pseudo-PDB scans to compute the new $max\{q'_{kj} \mid j \geqslant 1 \text{ and } j \neq i\}$ values, which is very costly.

However, for the proposed $\overline{aub}$ UB, if $\exists y = a_i \succ R$: $Ry \in [R]$ and $\overline{aub}(Ry) < mu$, then we can remove $Ry$ from the original equivalence class $[R]$ to obtain the new reduced class $[R]^* := [R] \setminus \{Ry\}$, or equivalently ignore (without having to remove) the $i^{th}$ column of the matrix $Q$ to obtain the corresponding reduced $Q^*$ (or $y$ is removed from PDB( $R$ )). Then, for any *high-* $\overline{aub}$ itemset $C = R \oplus E$ obtained by extending itemsets in the new $[R]$ equivalence class (i.e. $\overline{aub}(C) \geqslant mu$), the itemset $C$ does not contain $y$, values (in the *vertical* form) of $\overline{aub}(C)$ in $Q$ and $Q^*$ are the same, and $\overline{aub}(C) \leqslant rtub(C)$. In other words, $\overline{aub}(C)$ does not need to be updated in the reduced $Q^*$matrix but it is always tighter than the tightened $rtub$ UB of $auub$. This is one of the reasons why the dHAUIM algorithm, proposed in this paper, outperforms the state-of-the-art EHAUPM algorithm for mining HAUIs.

(*vi*). Note that, unlike the $\overline{\imath aub}$ and $\overline{laub}$ UBs, although $mau(A)$ is an UB of all its extensions $C = A \oplus B$ with $B \neq \emptyset$, i.e. $au(C) \leqslant mau(A)$ [20], $mau$ is not an UB on $au$, i.e. $\exists A$: $mau(A) < au(A)$. In general, $mau$ is incomparable with $\overline{\imath aub}$ and $\overline{laub}$. *Indeed*, the $auub$ values of five items $a$, $b$, $c$, $d$ and $e$ are respectively 616, 1120, 850, 856 and 1176. Thus, their ascending order $\prec$ w.r.t $auub$ is $a \prec c \prec d \prec b \prec e$. For $mu = 90$, the itemset $A = acb$, $au(A) = 290/3$ and $mau(A) = 74$, so $mau(A) < mu \leqslant au(A)$. Thus, ($mau(A) < mu$) cannot be used for pruning $branch(A)$. Furthermore, for $mu = 105$, branches starting from itemsets $A$ and $acde$ cannot be pruned by the $mau$ values of their prefixes because $mau(a) = 509.8 > mau(ac) = 554/3 > mau(acd) = 130.05 > mu$. Meanwhile, since $\overline{laub}(A) = 308/3 < mu$, $\overline{laub}(acde) = 38.25 < \overline{\imath aub}(acde) = 70 < mu$, $branch(A)$ and $branch(acde)$ are still pruned by $\overline{laub}$ and/or $\overline{\imath aub}$.

*Finally*, although $\overline{aub}_1$ and $auub$ are $\mathcal{AM}$, $\overline{aub}$ and $rtub$ have the same $\mathcal{BIDEAM}$ property, $\overline{laub}$ and $lub$ satisfy the same $\mathcal{DPC}$, however $\overline{laub}$, $\overline{aub}$ and $\overline{aub}_1$ are respectively tighter than $lub$, $rtub$ and $auub$. Thus, from a theoretical perspective, using the new $\overline{laub}$, $\overline{aub}$ and $\overline{aub}_1$ UBs for mining HAUIs will be certainly better than using the $lub$, $rtub$ and $auub$ UBs used in [18], [19]. In general, $\overline{laub}$, $\overline{\imath aub}$ and $mau$ are incomparable.

To summarize the above discussion, Table 4 compares the proposed $\overline{laub}$, $\overline{\imath aub}$, $\overline{aub}$, $\overline{aub}_1$ UBs and previous $lub$, $rtub$, $lubau$, $tubau$, $auub$, $mau$ UBs in terms of values and pruning effects.

Having shown that the proposed UBs have desirable properties, an interesting question arises, which is how to compute them efficiently. A key issue for computing the proposed UBs is how to quickly *calculate* the column vector $\mathcal{V}(C)$ of a child node $C$ using the vector $\mathcal{V}(P)$ of its parent $P$ in the prefix-tree. Assume that the itemset $C$ is obtained

Table 4. Comparison of values and pruning effects of UBs.

| Pruning Effect of UBs | $\overline{aub}_1 \ll auub$ | $\overline{aub} \ll rtub$ | $\overline{\imath aub} \ll \overline{aub}$ | $\overline{laub} \ll lub$ | $lubau$ | $tubau$, $mau$ |
|---|---|---|---|---|---|---|
| $\mathcal{AM}$ | ✓ | | | | | |
| $\mathcal{BIDEAM}$ | ✓ | ✓ | | | | |
| $\mathcal{FEAM}$ | ✓ | ✓ | ✓ | | ✓ | |
| $\mathcal{DPC}$ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

by joining two itemsets $LP = P'a_L$ and $RP = P'a_R$, where $P'$ is a common prefix and $a_L$, $a_R$ are items. If the QDB contains a large number of transactions, the sets $\rho(LP)$ and $\rho(RP)$ can be very large. Thus, calculating and storing the intersection $\rho(C) = \rho(LP) \cap \rho(RP)$ can consume much time and memory. To address this problem, the concept of diffset has been proposed in frequent itemset mining [21].

**Definition 7** (*Diffset of an itemset* [21]). *The diffset of the itemset $C$ is denoted as $d(C)$ and defined as $d(C) \triangleq \rho(LP) \setminus \rho(C)$, where "\" is the set difference operator. It was shown that if $C$ is a 1-itemset, then:*

$$d(C) \triangleq \mathcal{D} \setminus \rho(C), \quad (6)$$

*and for any k-itemset $C$ with $k \geqslant 2$, then*

$$d(C) = d(RP) \setminus d(LP). \quad (7)$$

Because the diffset $d(C)$ is always smaller than $\rho(LP)$ and $d(RP)$, it was shown that using $d(C)$ with formulas (6)-(7) instead of $\rho(C)$ to calculate the support of an itemset $C$ ( $supp(C) = supp(LP) - |d(C)|$, since $\rho(C) = \rho(LP) \setminus d(C)$) often allows to reduce memory usage and speed up calculations in frequent itemset mining [21]. The next paragraph explains how the concept of diffset is modified to be used for calculating UBs.

**Modified diffset technique for vertical UB calculations.**

By definition, the vector $\mathcal{V}(C)$ can be obtained using $\rho(C)$. Based on formulas 6 and 7, this article proposes two recursive formulas (Proposition 3) for quickly computing the vector $\mathcal{V}(C)$ of a child node $C$ based on the vector $\mathcal{V}(LP)$ and the small diffset $d(C)$ instead of $\rho(C)$.

**Proposition 3** (*Recursive formulas for fast calculating $\mathcal{V}(C)$*). *Assume that the itemset $C$ is an item-extension of a given itemset $P$. Thus, $minInd(P) = minInd(C)$. Based on $d(C)$ and $\mathcal{V}(P)$, the value of $\mathcal{V}(C) = \{v_j(C), j \geqslant minInd(C)\}$ is computed recursively as follows:*

$$v_j(\emptyset) = \sum_{i:\, q'_{ij} > 0} q'_{ij} = u(a_j), \forall j \geqslant 1, \quad (8)$$

$$v_j(C) = v_j(P) - \sum_{i:\, t_i \in d(C)} q'_{ij}, \forall j \geqslant minInd(C). \quad (9)$$

*In particular, if $d(C) = \emptyset$, then $\mathcal{V}(C) = \mathcal{V}(P)$.*

**Proof**. Since $C$ is an extension of $P$ with an item $a_R$, then $C = Pa_R$, and thus $minInd(P) = minInd(C)$. For any $j \geqslant minInd(P) = minInd(C)$, then $\rho(C) = \rho(P) \setminus d(C)$, $\rho(P)$ contains $\rho(C)$ and $d(C)$. Thus, $v_j(C) \triangleq \sum_{i:\, t_i \in \rho(C)} q'_{ij} = \sum_{i:\, t_i \in \rho(P)} q'_{ij} - \sum_{i:\, t_i \in d(C)} q'_{ij} = v_j(P) - \sum_{i:\, t_i \in d(C)} q'_{ij}$.

In particular, if $d(C) = \emptyset$, then $v_j(C) \equiv v_j(P)$, $\forall j \geqslant minInd(P) = minInd(C)$, i.e. $V(C) = V(P)$. □

In other words, $\mathcal{V}(C)$ can be calculated directly from $\mathcal{V}(P)$ and the rows $\{q'_{ij} \mid j \geqslant minInd(C)\}$ with $t_i \in d(C)$ in

the matrix $\mathcal{Q}$ corresponding to the diffset $d(C)$. Because diffsets are generally small, using diffsets is expected to reduce both runtime and memory consumption for computing $\mathcal{V}(C)$. Finally, based only on $\mathcal{V}(C)$, we can directly calculate the utility $u(C)$, and all the proposed upper-bounds $\overline{aub}_1(C)$, $\overline{aub}(C)$, $\overline{iaub}(C)$ and $\overline{laub}(C)$, using formulas (1) - (5) and $au(C) = u(C) / |C|$.

For example, consider the itemset $C = ad$, which is an extension of the prefix $LP = a$ obtained by combining $LP$ with an itemset $RP = d$ in the prefix tree. It is found that $\rho(LP) = 2456$, $d(LP) = 13$, $\rho(RP) = 12345$, $d(RP) = 6$, and $d(C) = d(RP) \setminus d(LP) = 6$. Thus $\rho(C) = \rho(LP) \setminus d(C) = 245$. Moreover, $\mathcal{V}(a) = \{v_j(a), j \geqslant 1\} = (90, 560, 70, 14, 16)$, $v_1(ad) = v_1(a) - \sum_{t_i \in d(C)} q'_{ij} = 90 - q'_{61} = 90 - 18 = 72$, and similarly $\mathcal{V}(ad) = (72, 240, 70, 14, 10)$. Thus, $u(ad) = v_1(ad) + v_4(ad) = 86$, $u(ad) = 43$, $\overline{aub}_1(ad) = \overline{aub}(ad) = 240$, $\overline{iaub}(ad) = 72$ and $\overline{laub}(C) = 53$. Now, consider another example. Let $C = acd$ be the itemset obtained by joining $LP = ac$ and $RP = ad$ (or the extension of $ac$ with $d$). We have that $d(c) = 26$, $d(ac) = d(c) \setminus d(a) = 26$, so $d(ac) \supseteq d(ad)$ and $d(acd) = d(ad) \setminus d(ac) = \emptyset$. Thus, $\mathcal{V}(acd) = \mathcal{V}(ac) = (66, 240, 70, 10, 7)$, $u(acd) = 146$, $u(acd) = 146/3$, $\overline{aub}_1(acd) = \overline{aub}(acd) = 240$, $\overline{iaub}(acd) = 70$ and $\overline{laub}(C) = 167/3$. It is observed that the cardinality of $d(C)$ is generally smaller than the one of $\rho(C)$, e.g. $|d(a)| = 2 < |\rho(a)| = 4$, $|d(d)| = 1 < |\rho(d)| = 5$, $|d(ad)| = 1 < |\rho(ad)| = 3$ and $|d(acd)| = 0 < |\rho(acd)| = 2$.

## 4 THE DHAUIM ALGORITHM

Based on the novel $\overline{aub}_1$, $\overline{aub}$, $\overline{iaub}$ and $\overline{laub}$ UBs and the recursive formulas of Proposition 3, this section presents an efficient algorithm, named dHAUIM (**d**iffset-based **H**igh **A**verage **U**tility **I**temset **M**iner), for mining the set of all high average-utility itemsets $HAUS \stackrel{\text{def}}{=} \{(C, au(C)) \mid au(C) \geqslant mu\}$. The proposal of this algorithm answers the third research question ($Q_3$). The pseudocode of the algorithm is shown in Fig. 2. During the mining process, $HAU$ candidate itemsets are stored in a prefix-tree using a novel structure named IDUL (Itemset-Diffset-UtilityList). This structure contains the information of a node $C$ of the form $(C, d(C), \mathcal{V}(C))$. Recall that the notation $[P]$ denotes the set of item-extensions of a parent node $P$.

The **dHAUIM** algorithm (Fig. 2) takes as input a QDB $\mathcal{D}$ and the $mu$ threshold. It first computes the integrated matrix $\mathcal{Q}_{nxm}$, $(\rho(a_j), j \in J)$ and the vectors $(d(a_j) \stackrel{\text{def}}{=} \mathcal{D} \setminus \rho(a_j), j \in J)$, $V(\emptyset) \stackrel{\text{def}}{=} (v_j(\emptyset) = u(a_j), \forall j \in J)$ using formulas (6) and (8) (line 1). Then, the algorithm calculates the vectors $\mathcal{V}(a_j) \stackrel{\text{def}}{=} (v_k(a_j) = v_k(\emptyset) - \sum_{i:\, t_i \in d(a_j)} q'_{ik}, k \in J)$ using formula (9) (line 2). Then, based on $\mathcal{V}(a_j)$ and by the formulas (2) - (5), the UBs $\overline{aub}_1(a_j) \stackrel{\text{def}}{=} max\{v_k(a_j), \forall k \geqslant 1\}$ (line 2) and $\overline{aub}(a_j) = \overline{iaub}(a_j) = max\{v_k(a_j) \mid k \geqslant j\}$, $\overline{laub}(a_j) \stackrel{\text{def}}{=} au(a_j) + max\{v_k(a_j) \mid k > j\}$, $\forall j \in J$. The set $[\emptyset]$ of all high–$\overline{aub}_1$ items is then obtained by applying the *strong width pruning strategy SWP* (line 3). Afterwards, the recursive procedure **HAU-Search**$([\emptyset], HAUS)$ is called (line 5). In terms of implementation, to reduce memory usage, the integrated QDB can be represented using well-known techniques for storing sparse matrices.

*HAUS* **dHAUIM**$(\mathcal{D}, mu)$
- *Input*: a QDB $\mathcal{D}$, the minimum AU threshold $mu$.
- *Output*: the set of high-average utility itemsets $HAUS$.
1. Create the integrated matrix $\mathcal{Q}_{nxm}$.
2. Scan $\mathcal{Q}_{nxm}$ once to calculate the vectors $\mathcal{V}(\emptyset)$ and $\mathcal{V}(a_j)$ for each $a_j \in \mathcal{A}$;
3. $[\emptyset] = \{(a_j, d(a_j), \mathcal{V}(a_j)) \mid a_j \in \mathcal{A}$ and $\overline{aub}_1(a_j) \geqslant mu\}$; //*strong width pruning*
4. $HAUS = \emptyset$;
5. **HAU-Search**$([\emptyset], HAUS)$;
6. **return** $HAUS$;

**Fig. 2.** The **dHAUIM** algorithm.

For instance, consider the integrated matrix of Table 3. For each item $a_j \in \mathcal{A}$, we calculate $\rho(a_j)$, $u(a_j)$ to obtain $\mathcal{V}(\emptyset) = (90, 1120, 320, 28, 31)$ as shown in Table 5. For example, $\rho(b) = 1356$, $v_2(\emptyset) \stackrel{\text{def}}{=} u(a_2 = b) = q'_{12} + q'_{32} + q'_{52} + q'_{62} = 160 + 400 + 240 + 320 = 1120$. Based on $\mathcal{V}(\emptyset)$ and formulas (6)-(9), (2)-(5), we can quickly calculate the vectors $\mathcal{V}(a_j)$ and the UB values $\overline{aub}_1(a_j)$, $\overline{aub}(a_j) = \overline{iaub}(a_j)$, $\overline{laub}(a_j)$, $\forall j \in J$. For instance, $\rho(a) = 2456$, $d(a) = \mathcal{D} \setminus \rho(a) = 13$. Hence, $\mathcal{V}(a) = (v_k(a) = v_k(\emptyset) - \sum_{i \in d(a) = \{1,3\}} q'_{ik}, k \geqslant 1) = (90 - 0 - 0, 1120 - 160 - 400, 320 - 100 - 150, 28 - 6 - 8, 31 - 5 - 10) = (90, 560, 70, 14, 16)$, $au(a) = v_1(a) = 90$ and $\overline{aub}(a) = \overline{iaub}(a) = \overline{aub}_1(a) = max\{v_k(a) \mid k \geqslant 1\} = 560$.

**HAU-Search**$([P], HAUS)$
- *Input*: the set $[P]$ of all item-extensions of $P$, the set $HAUS$.
- *Output*: the updated $HAUS$ set.

1. **if** $([P] \neq \emptyset)$ **then** {
2. **for** each $(C_i, d(C_i), \mathcal{V}(C_i))$ in $[P]$ **do** {
3. **if** ($\overline{iaub}(C_i) \geqslant mu$ or $\overline{laub}(C_i) \geqslant mu$) **then** { //*depth pruning*
4. **if** $(au(C_i) \geqslant mu)$ **then**
5. $HAUS$.Add$(C_i, au(C_i))$;
6. **if** $(|[P]| > 1)$ **then** {
7. $[C_i] = \emptyset$;
8. **for** each $(C_j, d(C_j), \mathcal{V}(C_j))$ in $[P]$, with $j > i$ **do** {
9. $E = C_i \cup C_j$; $d(E) = d(C_j) \setminus d(C_i)$;
10. **Calculate** $\mathcal{V}(E)$;
11. **if** $(\overline{aub}(E) \geqslant mu)$ **then** //*width pruning*
12. $[C_i] = [C_i] \cup \{(E, d(E), \mathcal{V}(E)\}$;
13. }
14. **HAU-Search**$([C_i], HAUS)$;
15. }
16. }
17. }
18. }
19. **return;**

**Fig. 3.** The **HAU-Search** procedure.

For a $k$-itemset where $k \geqslant 2$, $C = ac$, we have that $d(C) = d(c) \setminus d(a) = 26$, $v_1(ac) = v_1(a) - q'_{21} - q'_{61} = 90 - 6 - 18 = 66$ and similarly, $\mathcal{V}(ac) = (90 - 6 - 18, 560 - 320, 70, 14 - 4, 16 - 3 - 6) = (66, 240, 70, 10, 7)$, so $au(ac) = (v_1(ac) + v_3(ac)) / 2 = 68$, $\overline{aub}(ac) = \overline{aub}_1(ac) = max\{v_k(ac) \mid k \geqslant$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2833478, IEEE Transactions on Knowledge and Data Engineering



Table 5. The values $\rho(a_j)$, $d(a_j)$ and $u(a_j)$ in $\mathcal{V}(\emptyset)$.

| TID \ Item $a_j$ | $a$ | $b$ | $c$ | $d$ | $e$ |
|---|---|---|---|---|---|
| $\rho(a_j)$ | 2456 | 1356 | 1345 | 12345 | 123456 |
| $d(a_j)$ | 13 | 24 | 26 | 6 | $\emptyset$ |
| $\mathcal{V}(\emptyset)$ | 90 | 1120 | 320 | 28 | 31 |

$1\} = 240$, $\overline{iaub}(ac) = max\{v_k(ac) \mid k = 1, 3 \text{ or } k > 3\} = 70$ and $\overline{laub}(ac) = au(ac) + max\{v_k(ac) \mid k > 3\} = 78$. Consider another itemset $ae$. It is found that $d(ae) = d(e) \setminus d(a) = \emptyset$, and thus $\mathcal{V}(ae) = \mathcal{V}(a)$, so $au(ae) = (v_1(ae) + v_5(ae)) / 2 = 53$, $\overline{aub}(ae) = \overline{aub}_1(ae) = 560$, $\overline{iaub}(ae) = max\{v_k(ae) \mid k = 1, 5 \text{ or } k > 5\} = 90$ and $\overline{laub}(ae) = au(ae) + max\{v_k(ae) \mid k > 5\} = 53$.

The pseudocode of the **HAU-Search** procedure is shown in Fig. 3. It takes as input a set of item-extensions of a prefix $P$ and the current result set $HAUS$. For each child node $C_i$ in $[P]$, the *depth pruning strategy DP* is applied by simultaneously using two incomparable UBs $\overline{iaub}(C_i)$ and $\overline{laub}(C_i)$. If $\overline{iaub}(C_i) < mu$ or $\overline{laub}(C_i) < mu$, then the whole branch starting from $C_i$ is removed from the IDUL prefix tree (line 3). Otherwise, if $au(C_i) \geqslant mu$, then $C_i$ is added to the $HAUS$ set (lines 4-5). Next, in lines 6-10, $C_i$ is joined with each of its right siblings $C_j$ in the prefix-tree to create an extension itemset $E$, and the values $d(E)$ and $\mathcal{V}(E)$ are quickly calculated using formulas (7) and (9). For the extension itemset $E$, if $\overline{aub}(E) < mu$, then it is *pruned* by using the *width pruning strategy WP* without putting it into the child class $[C_i]$. Otherwise, $E$ is added to $[C_i]$ (lines 11-12). Afterwards, the HAU-Search procedure is recursively called for each child class $[C_i]$ (line 14). The HAU-Search procedure is stopped if the $[P]$ set received as parameter is empty (line 1).

Consider the running example with $mu = 250$. Because $\overline{aub}_1(a_j) > mu, \forall j \in J$, the procedure receives $[\emptyset] = \{(a, 13, (90, 560, 70, 14, 16)), (b, 24, (48, 1120, 270, 18, 27)), (c, 26, (66, 800, 320, 24, 22)), (d, 6, (72, 800, 320, 28, 25)), (e, \emptyset, (90, 1120, 320, 28, 31))\}$ as parameter. The final result is the set $HAUS = \{(ab, 304), (b, 1120), (bc, 535), (bd, 409), (be, 573.5), (bcd, 1088/3), (bce, 1091/3), (bcde, 1109/4), (bde, 839/3), (c, 320)\}$. The IDUL tree representing the search space is shown in Fig. 4. In this figure, for each node $A$, for the sake of brevity, the field $\mathcal{V}(A)$ is omitted, while the values $d(A)$, $\overline{aub}(A)$, $\overline{laub}(A)$, $\overline{iaub}(A)$ and $au(A)$ are represented as: $[A]^{\boldsymbol{d(A)},\overline{aub}(A),\overline{laub}(A)}_{\overline{iaub}(A),au(A)}$. Note that the dHAUIM algorithm does not need to store the whole tree in memory. Only the part of the IDUL tree required by the depth-first search is kept in memory at any given time.

**Remarks.** *In the proposed algorithm, one should not remove UBs or replace them with other UBs reviewed in this paper. If a strong UB is removed or a weaker UB is replaced with a stronger one, then the number of HAUI candidates can increase, and this will decrease dHAUIM's performance. Conversely, if a strong UB is replaced with a weaker one, it may result in missing some HAUIs in the result set $HAUS$.*

(*i*). For example, although the improved UB $\overline{iaub}$ is tighter than $\overline{aub}$, both have different pruning effects (DP and WP). Thus, using the width pruning condition "if $(\overline{aub}(E) \geqslant mu)$" at line 11 of the HAU-Search procedure is necessary. *Indeed*, for $\mu = mu = 250$, if the condition is removed, two redundant itemsets $ac$ and $ad$ will be kept in $[a]$. As a result, two additional redundant candidates $[abc]^{6,240,102}_{240<\mu,97}$ and $[abd]^{6,240,97}_{240<\mu,91}$ will need to be considered (when joining $ab$ with $ac$ and $ad$).

(*ii*). Similarly, if we replace the $\overline{iaub}$, $\overline{laub}$ UBs with the stronger $\overline{aub}$ in the depth pruning condition at line 3 of HAU-Search, the algorithm may consider much more unpromising candidate itemsets. *In fact*, for the IDUL tree in Fig. 4 and $mu = 200$, the algorithm would have to consider ten additional redundant candidates $A \in \{abc, abd, abcd, abce, abcde, abde, acd, ace, ade, acde\}$ having average-utility values $au(A)$ that are less than $mu$. On the contrary, if $\overline{iaub}$ or $\overline{laub}$ are used at line 3, then $\overline{aub}(A) \geqslant mu > \overline{iaub}(A)$, and these unpromising candidates are not considered.

(*iii*). Since the $\overline{iaub}$ and $\overline{laub}$ UBs do not satisfy the $BiDEAM$ property, we cannot replace the strong $\overline{aub}$ UB with weaker $\overline{iaub}$ or $\overline{laub}$ UBs in the width pruning condition at line 11 of HAU-Search. *Indeed*, assume that we replace the condition "if $(\overline{aub}(E) \geqslant mu)$" with "if $(\overline{iaub}(E) \geqslant mu$ and / or $\overline{laub}(E) \geqslant mu)$" (*) and that $mu = 80$. For the two nodes $[ab]^{24,560,324}_{560,304}$ and $[ac]^{26,240,78}_{70<mu,68}$, we have that $\overline{iaub}(ac) = 70 < \overline{laub}(ac) = 78 < mu$ and the modified condition (*) does not hold. Hence, $ac$ will not be in $[a]$ and thus the itemset $abc$ which is the union of $ab$ and $ac$ will be missing in the result set $HAUS$ (because $au(abc) = 290/3 > mu$).

## 5 Experimental Evaluation

To evaluate the proposed dHAUIM algorithm, extensive experiments were carried out on a computer equipped with an Intel(R) Core(TM) i5-6200U 2.3 GHz CPU with 4 GB of memory, running Windows 8.1. The performance of the proposed dHAUIM algorithm was compared with four state-of-the-art algorithms for mining high average utility itemsets, namely HAUI-Miner [17], EHAUPM [18], D-FHAUM [19] and MHAI [20]. All algorithms are implemented using the C# programming language.

In the experiments, three real-life datasets, namely Mushroom, Chess and Online_retail, and several synthetic datasets are used, where Mushroom and Chess can be obtained from [22] and Online_retail is available at the UCI machine learning repository (http://www.lsbu.ac.uk/). The datasets Mushroom and Chess contain actual transactions with synthetic utility values while Online_retail contains real utility values and transactions occurring between 2010 and 2011 for a UK-based and registered non-store online retail. Synthetic datasets have been generated using the IBM Quest data generator (obtained from [22]) with parameters described in Table 6. Note that transactions in synthetic databases contain items but no purchase quantities and utility values. Therefore, the simulation model described in [6] was used to generate these values. Characteristics of considered datasets are shown in Table 7.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2833478, IEEE Transactions on Knowledge and Data Engineering



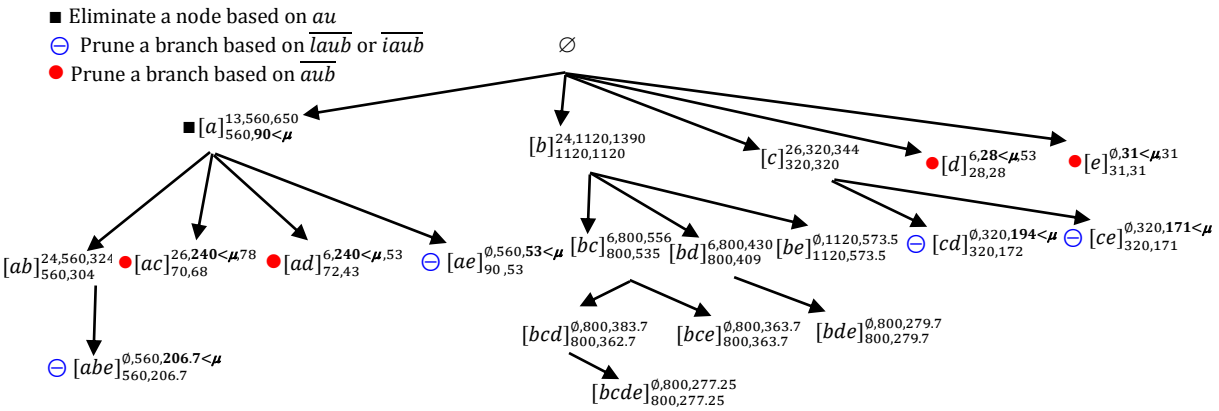


**Fig. 4** The IDUL tree for $\mu \stackrel{\text{def}}{=} mu = 250$, where each node is denoted as $[A]^{\boldsymbol{d}(A),\overline{\boldsymbol{aub}}(A),\overline{\boldsymbol{iaub}}(A)}_{\overline{\boldsymbol{iaub}}(A),\boldsymbol{au}(A)}$.

Table 6. Parameters of the IBM Quest Synthetic Data Generator.

| Parameters. | Meaning |
|---|---|
| T | Average transaction length |
| I | Average size of the maximal potentially frequent itemsets |
| N | Number of distinct items |
| D | Number of transactions (in thousands) in the dataset |

In the experiments, the *mu* threshold is expressed as a percentage of $TU$ (the total utility of the dataset, see Definition 2).

Table 7. Characteristics of datasets.

| Dataset | No. of trans. | No. of items | Average trans. length | Type of data |
|---|---|---|---|---|
| Mushroom | 8,124 | 119 | 23 | Real-life |
| Online_retail | 19,283 | 3785 | 19.9 | Real-life |
| Chess | 3,196 | 76 | 37 | Real-life |
| T11I6N30D100K | 100,000 | 30 | 11 | Synthetic |
| T15I9N100D100K | 100,000 | 100 | 15 | Synthetic |
| T20I9N50D100K | 100,000 | 50 | 20 | Synthetic |

### 5.1 Influence of the pruning strategies

As presented in Subsection 3.3, DP, WP and SWP are three novel strategies used by the dHAUIM algorithm to prune the search space efficiently. To evaluate their influence in dHAUIM, we considered two groups of strategies, depth pruning strategy (DP) and width pruning strategies (WP and SWP), and compared the runtimes of dHAUIM for three different cases: (1) dHAUIM with DP; (2) dHAUIM with WP and SWP; (3) dHAUIM with all DP, WP and SWP strategies (All). Fig. 5 shows the results for the real-life dataset Mushroom and the synthetic dataset T16I9N60D125K in terms of runtime.

As shown in this figure, for T16I9N60D125K, the pruning effect of DP is better than that of WP&SWP for the *mu* values of 0.3 and 0.4% but worse for the values of 0.6 and 0.7%. Meanwhile, on Mushroom, the runtimes of WP&SWP are always less than those of DP for all the tested *mu* values. This shows that although UB values of the strategy group WP&SWP are often larger than those in DP and the pruning conditions based on WP&SWP are less to be met, but their pruning effect is stronger than DP. The reason is that the WP&SWP strategies prune both backward and forward extension branches while DP eliminates only the forward extension branch. Note that when combining both the strategy groups, the runtimes of dHAUIM are reduced significantly, compared to using each of them separately. This demonstrates the importance of each of the novel strategies proposed in the dHAUIM algorithm.

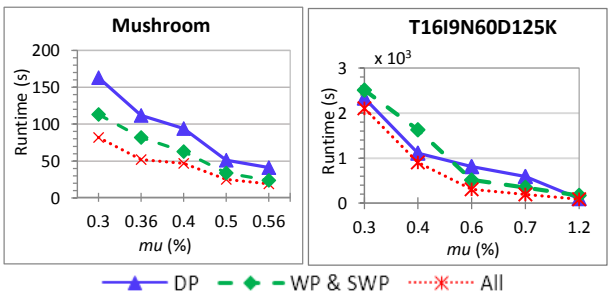


**Fig. 5** Runtimes of dHAUIM by using the pruning strategies

### 5.2 Influence of the *mu* parameter

The second experiment assesses the influence of the *mu* parameter on execution time and number of join operations for the datasets in Table 7. In this experiment, we use three real-life datasets. Mushroom and Chess have been used frequently to evaluate state-of-the-art algorithms and Online_retail contains real utility values. Also, we used synthetic datasets to vary characteristics such as the number of distinct items and the average length of transactions. In addition, various ranges of *mu* values have been tested on the datasets. These ranges either cover all the *mu* values which were employed to test the previous algorithms (on Mushroom and Chess) or cause large changes in terms of number of high average-utility itemsets and join operations (see Table 8 and Fig. 7). Note that, in Table 8, we only

Table 8. Number of high average-utility itemsets

| Dataset | *mu* (%) | | | | |
|---|---|---|---|---|---|
| Online_retail | *0.18* | *0.2* | *0.22* | *0.25* | *0.3* |
| #HAUIs | 122 | 101 | 82 | 55 | 31 |
| T20I9N50D100K | *1.2* | *1.6* | *1.9* | *2.0* | *2.3* |
| #HAUIs | 546 | 201 | 83 | 55 | 30 |

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2833478, IEEE Transactions on Knowledge and Data Engineering



introduce the number of high average-utility itemsets on T20I9N50D100K and Online retail due to the space limitation.

**Runtime**. The runtime of the proposed dHAUIM algorithm is compared with those of EHAUPM, D-FHAUM, MHAI and HAUI-Miner for various *mu* values on both real-life and synthetic datasets. Results are shown in Fig. 6. In the figure, we find that the runtimes of the algorithms decrease as *mu* is increased. The reason is that the numbers of HAUIs (#HAUIs) discovered and join operations performed naturally decrease as *mu* is increased (see Table 8 and Fig. 7).

It can be observed that dHAUIM is in general one to two orders of magnitude faster than the remaining algorithms for all *mu* values, and especially for low *mu* values. For example, on Chess for all considered *mu* values, dHAUIM is about 24 to 168 times, 2 to 43 times, 32 to 239 times and 5 to 37 times faster than EHAUPM, D-FHAUM, HAUI-Miner and MHAI, respectively. The reason for this is that, as discussed in Subsection 3.3, the three novel $\overline{laub}$, $\overline{aub}$ and $\overline{aub}_1$ upper-bounds are respectively tighter than $lub$, $rtub$ and $auub$, used by EHAUPM, D- FHAUM and HAUI-Miner. Upper-bounds greatly influence the performance of these algorithms. Besides, because $rtub$ relies on the horizontal database form, the EHAUPM algorithm often spends much time to recalculate or update $rtub$ values when items are removed from projected databases. Conversely, the dHAUIM algorithm computes $\overline{laub}$ using the vertical form. It thus only need to ignore columns in the database corresponding to items removed from the projected database. In addition, as mentioned in Subsection 3.3, although the $lubau$, $tubau$ and $mau$ upper-bounds [19], [20] are incomparable with the three novel $\overline{aub}_1$, $\overline{aub}$ and $\overline{iaub}$ upper-bounds, the amplitude of the values of $lubau$ and $tubau$ are quite large and $mau$ may not be an UB on $au$. Thus, candidate pruning strategies based on $lubau$ and $tubau$ are less stable in terms of efficiency and effectiveness. In general, the three proposed pruning strategies based on the novel $\overline{iaub}$, $\overline{laub}$, $\overline{aub}$ and $\overline{aub}_1$ upper-bounds allows the dHAUIM algorithm to prune a large number of unpromising candidates from the search space early. Consequently, the number of join operations performed by dHAUIM is much less than those of the compared algorithms (see Fig. 7), which leads to better performance in terms of runtime.

**Number of join operations**. The number of join operations performed by the algorithms was also recorded for all datasets and various *mu* values. Results are shown in Fig. 7. As it can be seen in this figure, the number of join operations of dHAUIM is much less than that of the previous algorithms MHAI, D-FHAUM, EHAUPM and HAUI-Miner. Thus, the search space of dHAUIM can be decreased dramatically. As a result, dHAUIM is also much faster than the other algorithms (see Fig. 6). For example, on T15I9N100D100K, when *mu* is increased from 0.42 to 0.9%, the number of join operations performed by dHAUIM is 69.4 to 98.8% less than the previous state-of-the-art algorithms.

### 5.3 Influence of database parameters

To evaluate the influence of database characteristics on the proposed algorithm's efficiency, a third experiment was performed where parameters of synthetic datasets (see Table 6) were varied.

In the following, a parameter letter followed by the wildcard (*) symbol indicates that the parameter is varied

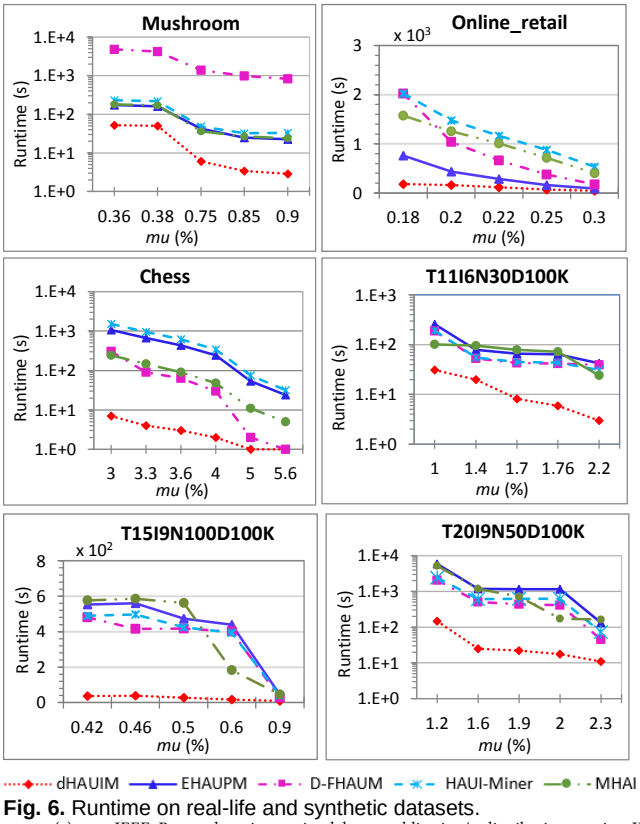


**Fig. 6.** Runtime on real-life and synthetic datasets.

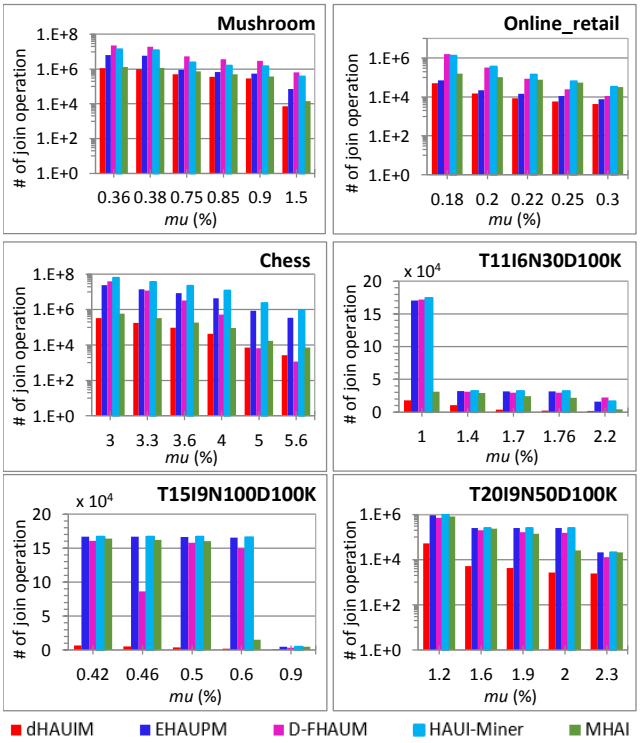


**Fig. 7.** Number of join operations on real-life and synthetic datasets.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2833478, IEEE Transactions on Knowledge and Data Engineering



for that dataset. The IBM Quest Synthetic Data generator was used to generate synthetic datasets for various parameter values. Fig. 8 shows the runtimes and number of join operations of the compared algorithms, respectively. In Fig. 8, it can be observed that when dataset parameters increase, dHAUIM shows linear scalability in terms of runtime, while EHAUPM, D-FHAUM, MHAI and HAUI-Miner are less stable and show considerable runtime fluctuations. In this experiment, dHAUIM always outperforms the other algorithms. For example, for the parameter N = 100 on the T16I9N*D100K database and D = 125K on T16I9N60D*K, dHAUIM is one to two orders of magnitude faster than EHAUPM, D-FHAUM, MHAI and HAUI-Miner. In terms of number of join operations, Fig. 8 shows that dHAUIM performs much less join operations than compared algorithms, especially for large database parameter values. This explains the small runtimes of the dHAUIM algorithm for mining the set of all HAUIs.

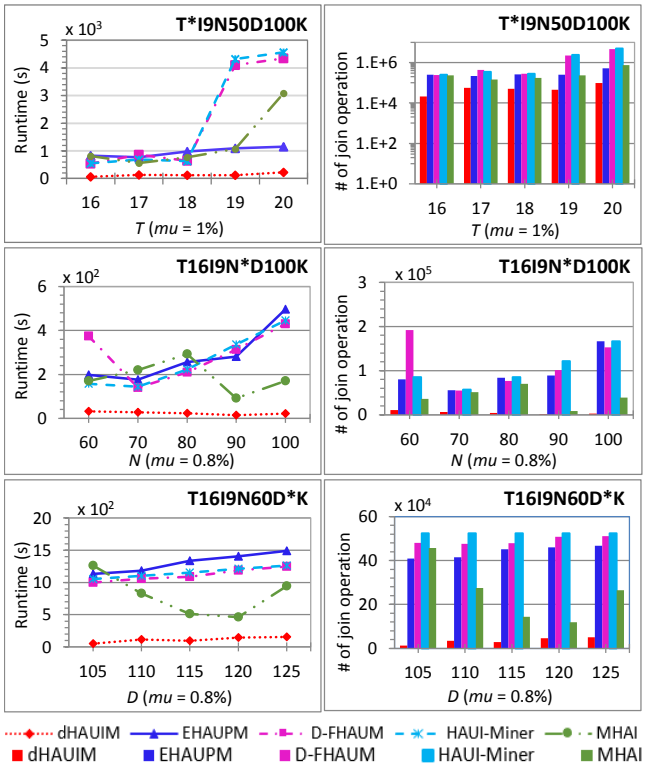


**Fig. 8.** Comparison of runtime and number of join operations w.r.t. database parameters.

Overall, the above experiments have shown that the proposed dHAUIM algorithm largely outperforms previous state-of-the-art algorithms in terms of runtime and number of join operations on both real-life and synthetic datasets, for various *mu* values. This shows that the proposed pruning strategies using the proposed upper-bounds are more effective than those of previous algorithms for mining all high average-utility itemsets.

## 6. Conclusion and Future Work

Based on the idea of computing utility values using a vertical form in quantitative databases, this paper has proposed four tight UBs, called $\overline{aub}_1$, $\overline{aub}$, $\overline{iaub}$ and $\overline{laub}$, a generic framework to evaluate UBs in terms of their pruning effects, and three pruning strategies to eliminate unpromising candidates early. A new IDUL tree structure was also developed to quickly calculate the average utility and UBs of itemsets using a recursive process. A novel algorithm named dHAUIM has been further presented to efficiently mine high average-utility itemsets. An extensive experimental evaluation was carried out. Results have shown that dHAUIM outperforms four state-of-the-art HAUI mining algorithms in terms of execution time and number of join operations on both real-life and synthetic databases.

Note that, the proposed generic framework to evaluate utility upper-bounds using anti-monotone-like criteria and the corresponding pruning strategies proposed in this study can be extended for the more general problem of mining all high utility sequences in quantitative sequence databases. This will be considered for future work.

## Acknowledgment

This work was supported by Vietnam's National Foundation for Science and Technology Development (NAFOSTED) under Grant Number 102.05-2017.300.

## References

[1] R. Agrawal, T. Imielinksi and A. Swami, "Mining Association Rules between Sets of Items in Large Database," in *The ACM SIGMOD International Conference on Management of Data*, 1993.

[2] V. Torres, K. Chiu and M. Vasudeva, "WFIM: weighted frequent itemset mining with a weight range and a minimum weight," in *The 2005 SIAM International Conference on Data Mining*, 2005.

[3] U. Yun, "Efficient mining of weighted interesting patterns with a strong weight and/or support afinity," *International journal of Information Sciences*, vol. 177, no. 17, pp. 3477-3499, 2007.

[4] G. Lan, T. Hong, H. Lee, S. Wang and C. Tsai, "Enhancing the Efficiency in mining Weighted Frequent Itemsets," in *IEEE International Conference on Systems*, 2013.

[5] C. Cai, A. Fu, C. Cheng and W. Kwong, "Mining Association Rules with Weighted Items," in *The International Database Engineering and Applications Symposium*, 1998.

[6] Y. Liu, W. Liao and A. Choudhary, "A fast high utility itemsets mining algorithm," in *The Utility-Based Data Mining Workshop*, 2005.

[7] G. Lan, T. Hong and V. Tseng, "An efficient projection-based indexing approach for mining high utility itemsets," *J. Knowl. Inform. Syst.*, vol. 38, no. 1, pp. 85-107, 2013.

[8] M. Liu and J. Qu, "Mining high utility itemsets without candidate generation," in *ACM International Conference on Information and Knowledge Management*, 2012.

[9] H. Yao, H. Hamilton and C. Butz, "A foundational approach to mining itemset utilities from databases," in *The 2004 SIAM International Conference on Data Mining*, 2004.

[10] C. Ahmed, S. Tanbeer, B. Jeong and Y. Lee, "Efficient tree structures for high utility pattern mining in incremental databases," *J. IEEE Trans. Knowl. Data Eng,* vol. 21, pp. 1708-1721, 2009.

[11] R. Chan, Q. Yang and Y. Shen, "Minging high utility itemsets," in *IEEE International Conference on Data Mining*, 2003.

[12] T. Hong, C. Lee and S. Wang, "Mining High Average-Utility Itemsets," in *The 2009 IEEE International Conference on Systems, Man, and Cybernetics*, 2009.

[13] J.-W. Lin, T. Hong and W. Lu, "Efficiently mining high average utility itemsets with a tree structure," in *Lect. Notes Comput. Sci. 5990*, 2010.

[14] T. Hong, C. Lee and S. Wang, "Effective utility mining with the measure of average utility," *The international journal of Expert Systems with Applications,* vol. 38, no. 7, pp. 8259-8265, 2011.

[15] T. Lu, B. Vo, H. Nguyen and T. Hong, "A new method for mining high average utility itemsets," in *Lect. Notes Comput. Sci. 8838*, 2014.

[16] G. Lan, T. Hong and V. Tseng, "Efficiently mining high average-utility itemsets with an improved upper-bound strategy," *International Journal of Information Technology & Decision Making,* vol. 11, no. 5, pp. 1009-1030, 2012.

[17] J.-W. Lin, T. Li, P. Fournier-Viger, T. Hong and J. Zhan, "An efficient algorithm to mine high average-utility itemsets," *International journal of Advanced Engineering Informatics,* vol. 30, no. 2, pp. 233-243, April 2016.

[18] J.-W. Lin, S. Ren, P. Fournier-Viger and T. Hong, "EHAUPM: Efficient High Average-Utility Pattern Mining with Tighter Upper-Bound Models," *J. IEEE ACCESS,* vol. 5, pp. 12927 - 12940, 2017a.

[19] J.-W. Lin, S. Ren, P. Fournier-Viger, T.-P. Hong, J. Su and B. Vo, "A fast algorithm for mining high average-utility itemsets," *Applied Intelligence,* vol. 47, no. 2, pp. 331-346, 2017b.

[20] U. Yun and D. Kim, "Mining of high average-utility itemsets using novel list structure and pruning strategy," *Future Generation Computer Systems,* pp. 346-360, 2016.

[21] M. Zaki and K. Gouda, "Fast vertical mining using diffsets," in *The 9th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2003.

[22] P. Fournier-Viger, A. Gomariz, T. Gueniche, A. Soltani and C. Wu, "SPMF: a Java Open-Source Pattern Mining Library," *J. Machine Learning Research,* vol. 15, no. 1, pp. 3389-3393, 2014.

**Tin Truong** is a professor at the Department of Mathematics and Computer Science, University of Dalat, Vietnam. He received his B.S. degree in Mathematics from Dalat University in 1983 and his Ph.D. in Stochastic Optimal Control in 1990 at the Vietnam National University, VNU-Hanoi, Vietnam. His current research interests are artificial intelligence and data mining.

**Hai Duong** is a Ph.D. student of Computer Science at the University of Science—VNU-HCMC, Vietnam. His research interest is data mining. He received his M.S. degree in Computer Science in 2009 at the University of Science—VNU-HCMC, Vietnam, and he is currently a lecturer and a researcher at the Department of Mathematics and Computer Science, University of Dalat, Vietnam.

**Bac Le** received his B.S. degree in Mathematics in 1984 and the M.S. degree in Computer Sciences in 1990. He received his Ph.D. in Mathematics for Computers and Computing Systems in 2000 at the University of Science—VNU-HCMC, Vietnam. He is currently the Head of Department of Computer Science as well as the Vice Dean, Faculty of Information Technology. His current research interests include artificial intelligence, soft computing, data mining and data science.

**Philippe Fournier-Viger** is a full professor at the School of Natural Sciences and Humanities, Harbin Institute of Technology (Shenzhen), China, and adjunct professor at the Department of Computer Science, University of Moncton, Canada. He received the Ph.D. in Computer Science, at the University of Quebec in Montreal, Canada in 2010. He is the founder of the popular SPMF data mining library. His research interests include data mining, pattern mining, sequence mining, sequence prediction, cognitive modeling and real-life applications in industrial design.